Parameter-Free Spatial and Stream Mining

Spiros Papadimitriou

September 2005
CMU-CS-05-170

School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213

Thesis Committee:
Christos Faloutsos, Chair
Anastassia Ailamaki,
Phillip B. Gibbons,
Jiawei Han, University of Illinois, Urbana-Champaign

Submitted in partial fulfillment of the requirements
for the degree of Doctor of Philosophy

This material is based upon work supported by the National Science Foundation under Grants No. IIS-9817496, IIS-9988876, IIS-0113089, IIS-0083148, IIS-0209107, IIS-0205224, SENSOR-0329549, EF-0331657, IIS-0326322, NASA Grant AIST-QRS-04-3031, by the Pennsylvania Infrastructure Technology Alliance (PITA) Grant No. 22-901-0001, by the Defense Advanced Research Projects Agency under Contract No. N66001-00-1-8936, and by a generous fellowship from the Siebel Scholars Program. This work is supported in part by the Pennsylvania Infrastructure Technology Alliance (PITA), a partnership of Carnegie Mellon, Lehigh University and the Commonwealth of Pennsylvania's Department of Community and Economic Development (DCED). Additional funding was provided by donations from Intel, and by a gift from Northrop-Grumman Corporation.

Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation, DARPA, or other funding parties.

Keywords: Data mining, Outliers, Time series, Streams, Clustering

Dedicated to my parents
Στους γονείς μου, Κώστα και Θωμαή

"Ordnung ist heutzutage meistens dort, wo nichts ist.
Es ist eine Mangelerscheinung."

— *Bertold Brecht*[1]

[1]Quote taken from the introduction to Paul Feyerabend's "*Against Method*"

Abstract

Data mining is the extraction of knowledge from large amounts of data [HK01] and brings together the fields of databases, machine learning and statistics. Technological advances have enabled the creation of huge data repositories. The need to turn such data into useful knowledge has fueled the development of data mining techniques. From a database perspective, the emphasis is often placed on *scalability* and *efficiency*. Practical approaches can afford only a single or, at best, a few passes over the data, i.e., the algorithmic complexity must be linear with respect to dataset size.

Recently, in addition to the *data warehouse* model where data from multiple sources are integrated into a large store, the *streaming model* is emerging as an alternative data processing paradigm. Several applications produce a continuous stream of data (e.g., phone call detail records, web clickstreams or sensor measurements) that is too large to store in its entirety. Therefore, in stream mining we are allowed only a single pass over the data, without random access. Upon arrival of new observations, we have to incrementally update the data model. Furthermore, space complexity must be sublinear with respect to dataset size.

In this thesis we develop spatial and stream mining tools for discovery of interesting patterns. These patterns summarise the data, enable forecasting of future trends and spotting of anomalies or outliers. Beyond the emphasis on efficiency and scalability, we focus on simplifying or eliminating user intervention. Data mining algorithms must make the discovery task easy for average users. Unfortunately, many of the existing techniques require non-trivial user intervention at several steps of the process. Eliminating the requirement for user intervention should be a top priority in designing data mining methods.

We show that *multi-resolution* analysis (i.e., examining the data at multiple resolutions or scales) is a powerful tool towards these goals. In particular, for spatial data we employ the *correlation integral*. For time series streams we use the *wavelet transform* and related techniques. Furthermore, we leverage tools from signal processing (again wavelets and, also, *subspace tracking* algorithms) to extract patterns from streams. Finally, we also employ compression principles coupled with multi-level partitionings to automatically cluster spatial

data.

The first two parts of this thesis focus on spatial mining methods. In the first part we examine homogeneous spatial data, where all points belong to one class. In the second part we examine heterogeneous spatial data, where the points may belong to two or more different classe (e.g., species, galaxy types, etc). Finally, in the third part we focus on numerical, time series streams and mining techniques for both single and multiple streams.

Acknowledgements

When you set out on your journey to Ithaca,
pray that the road is long,
full of adventure, full of knowledge. [...]
Ithaca has given you the beautiful voyage.
Without her you would have never set out on the road.
She has nothing more to give you.
— Constantine P. Cavafy, "Ithaca"

This thesis is the end product of several years, but the events leading to it and the people involved are, perhaps, at least as important.

I am deeply grateful to my adviser, Christos Faloutsos. Christos provided much more freedom than anyone could ever ask or even hope for. He was always there to listen and to share (but never force) his opinions and experience on all subjects. He has offered advice on which classes to take (or, sometimes, not to take), on how to do research, write papers and give talks, but also on choosing careers or even choosing suits and ties (although the latter was with the help of his wife, Christina, who I would also like to thank—Christos may be good at many things, but fashion is not one of them). His advice has been valuable, even though I may not have always followed it. Christos has been a point of reference throughout my course of graduate studies, a role model as a researcher, a mentor and, most of all, as a person. All of us who have had the privilege to work with Christos and experience his contagious enthusiasm and optimism are indeed very fortunate.

I would also like to thank my committee members, Anastassia Ailamaki, Jiawei Han and Phillip B. Gibbons for their feedback and support.

I would also like to thank all of my colleagues and coauthors, for their constant stimulation, valuable feedback and discussions, both those whose work was involved with and contributed to this thesis (Anthony Brockwell, Deepayan Chakrabarti, Phillip B. Gibbons, Aristides Gionis, Alexander Hinneburg, Hiroyuki Kitagawa, Heikki Mannila, Dharmendra

Modha, Jimeng Sun, Panayiotis Tsaparas and Cui Zhu) as well as those who worked on other topics (Rakesh Agrawal, Roberto Bayardo, Daniel Gruhl, Kyriakos Mouratidis, Dimitris Papadias, Yasushi Sakurai, Agma Traina, Caetano Traina and Mengzhi Wang).

Natassa Ailamaki joined Carnegie Mellon at about the same time I started work on this research area and, over the years, has done more than her "fair share" by providing sometimes harsh but always friendly advice on a broad range of topics, ranging from presentation skills to classroom instruction. Also, Chris Olston gave me some valuable criticism and feedback during a practice talk, at the very last stages of this work.

Many people made my years at Carnegie Mellon and Pittsburgh pleasant and memorable. Among these, Stavros Harizopoulos, Hyang-Ah Kim, Yiannis Koutis and Stratos Papadomanolakis deserve a very special mention. If I were to list just the most memorable moments with them, I am sure this section would grow longer than the rest of this thesis. I owe both my sanity and my insanity to each of them, not necessarily in equal proportions.

Even though they did not have to abide me on an almost daily basis, I owe a special thanks to Deepay Chakrabarti, Nikos Hardavellas, Ioanna Pagani, Vlad Shkapenyuk and Jimeng Sun. With Jimeng and Deepay, also Christos's students, we shared part of the journey, even though each of us was his own captain. I will also not forget all the discussions we had with Jamie Gruzska, over countless emails and occasionally a few beers. Even though I do not spend much time in a wet darkroom anymore, the lessons learnt through our interactions extend well beyond photography.

Finally, my experience at Carnegie Mellon was made more pleasant and interesting by many other people at CMU and the database group members, including but not limited to: Umut Acar, Christoffer Hall-Fredriksen, Jure Leskovec, Dimitris Margaritis, Jia-Yu Pan, George Sapountzis, Bianca Schröder, Minglong Shao, Mengzhi Wang, Leejay Wu and Hyungjeong Yang.

I was also very fortunate to have the opportunity to visit and work at several places beyond Carnegie Mellon. This allowed me to explore different problems and, perhaps more importantly, to enrich my experience with different environments and cultures. Once again, I must thank Christos for allowing me to do this.

The Basic Research Unit (BRU) at HIIT in Finland is a stimulating environment, which attracts people from several disciplines and countries. The two months I spent there were very fruitful and I am indebted to Heikki Mannila for hosting me. I am also happy to have been part of a team with Aris Gionis, Evimaria Terzi and Panayiotis Tsaparas, both inside and outside of work. I also had a very pleasant collaboration with Alex Hinneburg, who

was visiting BRU at the same time. Finally, I'd like to thank my officemate Ydo Wexler for putting up with all the empty Coke bottles.

I am also grateful to Dimitris Papadias for hosting me at the University of Science and Technology (UST) in Hong Kong. Dimitris has a unique perspective which, in retrospect, taught me a lot. Kyriakos Mouratidis made my stay there even more interesting.

I spent a good six months as an intern at IBM Almaden, at a juncture in my graduate studies. This was back in 2000, at a time when I decided to make a turn-around and completely change research areas. I was indeed very fortunate to be in Rakesh Agrawal's group; Rakesh was a source of inspiration and solid, frank advice. It was also a true pleasure to work with Roberto Bayardo and Dan Gruhl; they made everything so smooth and apparently effortless that my time there never felt like "work." Beyond all matters technical, I would like to thank Dan also for loaning me his mountain bike and pushing me up those hills in Almaden, but turning me back when I "turned greenish-yellow" (although, to this day, I disagree that I ever was any shade of green or of yellow). My time at Almaden was made much more pleasant thanks to the other interns and IBMers, including Karin Cheung, Amit Somani, Ramakrishan Srikant, Yirong Xu and Yinghui Yang. I would also like to thank Dharmendra Modha, whom I met more recently (during Christos's sabbatical at Almaden), for his unique and maybe somewhat "poetic" perspectives on several topics, during our brief encounters. Finally, I would like to thank Yiannis Mavroidis. I have known him since our college years in Crete, where we spent several hours together, both inside and outside classes. I never managed to match his speed and coordination at Warcraft (but then again, neither did anybody else). He also hosted me at his place in Berkeley when I first arrived in California and had nowhere to stay.

The Computer Science Department and Carnegie Mellon has been an excellent environment, which has supported me throughout the course of this work. I would like to thank all of my teachers, in several topics and disciplines for their stimulation. I would also like to thank all those who make the gears run smoothly, including Sharon Burks, Catherine Copetas, Maurelle Copeland and Denny Marous.

I would also like to thank Becky Buchheit for her help with the automobile traffic datasets, Orna Raz for collecting the river gauge data, Mike Bigrigg for the temperature sensor data and Jeanne Van Briessen for the water distribution network data.

From my college years, I would like to thank my professors, including George Georgakopoulos and particularly Manolis Katevenis. The Computer Science Department in Heraclion was unique among Greek universities in many ways and it gave me a solid foundation for

my later endeavours. I will also fondly remember the times we spent there with Antonis, Fragiskos, Ilias, Kostas and Thanassis.

Going further back, there are several people who had a significant influence and it is not possible to list them all. However, Takis Moraitis and Lambros Stasis deserve special mention. Both were the best teachers and also friends one could have.

Most of all, I am grateful to my parents, Kostas and Thomais, without whose constant love and support none of this would have been possible.

Spiros Papadimitriou
September 29, 2005

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Data mining is the natural evolution of information technology and driven by a data-rich but information-poor situation [HK01]. The incorporation of digital computers in most aspects financial, industrial, scientific and even daily activities has led to the collection of large amounts of data. For example, retailers store details on each sales transaction (such as list of items purchased), long-distance service providers store call detail records for each phone call [GKMS01], astrophysicists collect numerous information (such as position and absolute brightness) for several hundred million celestial objects [SDS], and so on.

In all the above cases, individual data records have limited usefulness. However, when the data are viewed in the right way and at the appropriate level of detail (neither of which are known in advance), we may be able to extract informative patterns. For example, the fact that a customer bought milk last Monday at 6:20pm may be of limited use. However, the fact that 65% of the customers who bought milk also bought cookies may, for example, help in choosing product placement. Or, the fact that a particular elliptical galaxy with luminosity 5.75×10^{27} Watts is present at specific coordinates may not be very interesting. However, the fact that elliptical and spiral galaxies generally do not occupy the same region in the sky (within, say, a fraction of a degree) may help in explaining or verifying physical phenomena.

Development of data mining has been driven by the desire and need to extract concise and informative nuggets of knowledge out of these vast stores of raw data. Thus, data mining is the adaptation of machine learning and statistical processing techniques to very large collections of data. This adaptation usually relies on effectively exploiting and incorporating database technologies and methods.

In contrast to traditional database and information processing systems, data mining methods seek to discover patterns and trends in the data, as opposed to answering specific user queries. For example, a traditional database system is designed to answer queries

such as "what is the average population density in New York?" or "what was the average temperature in Pittsburgh?" Data mining seeks to find interesting pieces of information, such as "the average population density in Manhattan is significantly higher than the state average" or "tomorrow's temperature in Pittsburgh will be about 82°F (28°C)."

In contrast to traditional statistics and machine learning, data mining techniques place the most emphasis on *scalability* to large datasets (in the order of gigabytes or terabytes of data), rather than theoretical issues and properties (such as *consistency*, *bias* and *statistical efficiency*). However, machine learning also places an emphasis on *computational efficiency*, but to a lesser extent, at least traditionally. In fact, practical data mining approaches can afford only a single or, at best, a few passes over the data, i.e., the algorithmic complexity must be linear with respect to dataset size.

In addition, applications may also generate data in the form of *streams*. In this setting, massive amounts of data are produced and traditional systems cannot store and process them sufficiently fast. In fact, in some cases it may not be desirable or even feasible to store all the data. For example, network operators may collect information for each data packet that travels across the network. With billions of packets transmitted daily, the amount of such data can potentially be vast. However, end users or higher level applications still wish to extract useful information and patterns, often requiring immediate responses and unable to afford any post-processing. The streaming data model may be viewed as a generalisation of the traditional data warehouse model when the dataset size grows to infinity.

In the most general case, each individual data record (whether in a streaming setting or not) may be a tuple with fields of arbitrary data types (e.g., binary, numerical or string). We focus primarily on numerical values. Thus, datasets may be either a static collection of multi-dimensional vectors or infinitely growing time series. In this thesis, we will use the terms *spatial data* and *stream data* to refer to each of these cases, respectively.

At a very high level, there are several possible data mining tasks. In this thesis we focus on unsupervised methods and primarily on clustering and outlier detection. For time series streams we also examine forecasting models. These tasks are frequently inter-related, as explained in Section 1.2.

Finally, algorithms for discovering concise and informative patterns are unavoidably based on some prior notion of what to look for. In other words, starting from some set of models that can concisely describe the data, an algorithm will search for the best model. The family of models may be very broad and, consequently, allow discovery of several interesting, non-trivial patterns, so this prior "bias" is not as restrictive as it may sound. However, some data mining algorithms require the user to provide some non-trivial parameters that,

in some way, guide the search for the best model. Example of such parameters may be the number of clusters, the range of distance or time scales at which to examine the data, etc. This information is typically not known in advance. Data mining algorithms must make the discovery task easy for average users. We want to design methods that do not expose the users to any such parameters, even though the prior "bias" is impossible to eliminate. The search should return some patterns without *requiring* any guidance of this form.

1.1 Our contributions

In this thesis we develop tools for spatial and stream mining that satisfy the following requirements:

- Parameter free: Our methods should be able to search the space of patterns without requiring human intervention and guidance.
- Expressive: Our models should provide concise, powerful and intuitively interpretable patterns.
- Scalable and any-time: Our algorithms should scale to very large datasets, which practically means that they should require only one pass over the data. In the case of streams, we have to incrementally update the models and be able to provide up-to-date patterns instantaneously, while using limited memory.

This thesis focuses on the tasks of outlier detection, as well as clustering and forecasting. In the following sections, we elaborate on these tasks, on the broad range of data models we assume and on our proposed methods.

1.2 What: models, forecasts and outliers

Next, we discuss the main data mining tasks we focus on. In this thesis, we consider clustering and forecasting and also forecasting, which is, in a sense, the "dual" of the first two problems. First we give some informal definitions of these tasks.

Definition 1 (Clustering). *Given a set of data points, partition them into groups such that "similar" points are placed in the same group and "dissimilar" points are placed in different groups.*

In other words, clustering seeks to find groups of objects based on their "similarity," appropriately defined. Such a group of similar objects (i.e., a *cluster*) provides insights about patterns present in the data, in terms of their spatial properties. Each cluster is essentially a pattern. For example, the size of a cluster, the features that account for the similarity of its members and the extent of the cluster in each of the feature dimensions may be elements of such a pattern. The set of clusters also also provides useful information (e.g., how many such patterns are present in the dataset).

Definition 2 (Forecasting). *Given a sequence of data points, try to predict the next one or more points.*

Forecasting seeks to find patterns along the time dimension, as opposed to the feature or spatial dimension. Thus, forecasting models also incorporate the "arrow of time" and are typically *generative* (as opposed to *descriptive*).

Definition 3 (Outlier detection). *Given either an (unordered) set or a sequence of points, try to find those that deviate from "normal" (i.e., typical) behaviour.*

Clustering and forecasting seek to find the "normal" spatial patterns and trends, respectively. Hence, outlier detection may be viewed as the "dual" problem: points that deviate from the typical spatial distribution or temporal trends are *outliers*.

1.3 How: parameters

As explained before, any algorithm for discovering concise and informative patterns is unavoidably based on some prior notion of what to look for. Designing a data mining method requires striking the appropriate balance, so as to achieve the following goals:

- Model design: A model should be both powerful and concise. These two desiderata are partly conflicting. It is easy to create models that have high descriptive power, at the expense of description length and vice versa.
- Algorithm design: The models should admit search strategies that do not require user guidance in the form of data dependent parameters.

Since data mining algorithms must make the discovery task easy for average users and allow the extraction of information that was previously unknown, a top priority in designing data mining methods should be to eliminate, as much as possible, the requirement for user intervention to guide the search for patterns.

Our main approach to deal with is that any parameters involved are typically data driven, as opposed to data dependent. Hence, the output of our algorithms is less sensitive to these parameters in the sense that they typically produce some meaningful results regardless of the chosen values and parameter defaults are typically sufficient. As a simple illustrative example, consider the problem of detecting outliers, based on some predefined notion of pairwise "similarity" or distance. One possible scheme would be to detect as outliers those points p for which at least $F\%$ of the remaining points are at a distance D or more from p. However, the parameter D is data dependent and their choice depends essentially on the distribution of pairwise distances. A better approach might be to choose a distance threshold relative to, say, the mean pairwise distance (as opposed to an absolute distance value). Thus, we may instead flag as outliers those points for which $F\%$ of the points are at a distance larger than $\mu_D + k\sigma_D$, where μ_D and σ_D are the average pairwise distance and its standard deviation. We still have the parameter k (instead of D), but the outlier flagging criterion is now data driven, as opposed to data dependent. Of course, this scheme is very simple and suffers from other problems such as, e.g., local variations in the pairwise distance distribution (see also Chapter 3). This is because the data model is very simple: we essentially assume that all points are drawn from a single distribution. However, by choosing an appropriate data model and by framing the problem so that any parameters are data driven, we can get high-quality results with no user intervention. Finally, we also employ principles from information theory and compression in order to choose both the best model (i.e., parameters) as well as the specific parameter values (see also Chapter 9).

1.4 Spatial data

In this section we discuss the spatial data models considered in this thesis. In this thesis we consider mainly pure spatial data, as well as extended spatial data where each point may also have non-spatial attributes or features. First, we give the high-level definitions of the data models we assume, in their most general form.

Definition 4 (Spatial data model). *The dataset D consists of d-dimensional vectors, i.e., it is of the form $D := \{x_1, x_2, x_3, \ldots, x_n\}$, with $x_i \in \mathbb{R}^d$, for $1 \leq i \leq n$.*

For spatial data, we consider the problems of outlier detection (Chapter 3 and 4), as well as the "dual" of clustering (Chapter 5).

Definition 5 (Extended spatial model). *Each item in the dataset D is a d-dimensional vector*

together with m binary attributes (or features). Formally, the dataset D is of the form $D := \{(x_1, a_1), (x_2, a_2), \ldots, (x_n, a_n)\}$, *with* $x_i \in \mathbb{R}^d$ *and* $a_i \in \{0,1\}^m$, *for* $1 \leq i \leq n$.

We will consider the case $m = 1$ for outlier detection (Chapter 8). For clustering, we consider the problem of *simultaneously* grouping the points and the binary attribute fields (Chapter 9). In the latter case, we restrict the problem to raster data, where essentially each data point x_i belongs to $\mathbb{N}^d$, instead of $\mathbb{R}^d$ as is the case for arbitrary vector data. Consequently, we employ techniques from image compression to group both the pixels (i.e., points) as well as binary attributes. Note that, from a general vector data set we may construct a raster data set by appropriately discretising the space (for example, substitute geographical longitude and latitude with city blocks).

1.4.1 Example domains and applications

In this section we describe briefly application domains where spatial data arise and give some illustrative, intuitive examples.

Biodiversity and geographical data In a number of applications, the numerical coordinates are low-dimensional and correspond to geographical location.

- Such data are collected for ecosystems and other natural systems, for example chimpanzee locations painstakingly collected over several decades [Goo84], or presence of several hundreds of bird species over a large area [LMP03], etc.

- In geographical/geopolitical applications, we may have points that represent populations, land and water features, regional boundaries, retail locations, police stations, crime incidence and so on.

These data are particularly rich and a number of analyses are possible. For example, some interesting patterns are:

- Most chimpanzees tend to cluster together and occupy the same locations, except for a few isolated individuals which may merit special study.

- Houses and retail stores generally tend to occupy different areas, with few exceptions (e.g., large metropolitan areas). Such information can help, for example, in city planning.

- The relative density of police stations with respect to either houses or crime incidence is constant, with the exception of some locations. This may indicate, e.g., poor coverage at some locations (say, if the relative density of stations is lower), or other factors that affect crime rate (say, if the relative density of crime incidence is higher).

Astrophysics data Astrophysicists and astronomers catalogue the location and absolute intensity of celestial objects, such as galaxies. Viewed abstractly, this type of data is similar to geographical data. However, the scale of ongoing cataloguing efforts is very large [SDS]. Furthermore, even single deviant observations would potentially be of great interest; it is well known that the distributions of different celestial objects follow certain laws and any violations would indicate either interesting phenomena or data acquisition errors. Finally, the dimensionality of the data may be quite large (except coordinates and intensity, various other properties are often measured, such as redshift,). For all these reasons, this type of data deserves special attention. Interesting patterns may be, for example:

- Elliptical and exponential galaxies form small clusters of one type and these clusters "repel" each other.
- In the extended space of all numerical properties, celestial objects tend to occupy only certain subspaces.

User profile data In several cases (e.g., retail, websites [ABKS99], network operators) user profile data are routinely collected. Discovering clusters and outliers in such data would help in tasks such as website design, product targeting, etc.

Other In a number of cases, data points may not necessarily belong to a multi-dimensional Euclidean space or even to a vector space. However, in several cases, a notion of *object similarity* can be established, by means of a distance function that often satisfies the triangle inequality. Such data *could* potentially be embedded in a vector space, but that may be time consuming. Our basic definitions apply in the case of metric spaces as well and our methods can be extended to handle such cases. Some examples of such data and application domains are:

- Document collections, with distances such as cosine similarity.
- Images and in general multimedia data, which in raw form may be treated as vectors but, often, a better approach is to represent them via a smaller set of extracted *features* (such as average intensity, colour histograms, texture information etc).

- Graphs, with similarity established via graph kernels or other means.

1.5 Stream data

In this section, we discuss the stream data model considered in this thesis. In general, individual records generated by a stream may contain values of arbitrary types (e.g., numbers—such as temperature, network packet counts, stock prices, etc.—or strings—such as URLs visited). In this thesis, we consider numerical streams, both single (equivalently, scalar-valued time series) as well as multiple (equivalently, vector-valued time series). Essentially, in contrast to spatial data, the data set size grows without bound. Thus, for our investigations, the stream data model we consider is the following.

Definition 6 (Stream data model). *The dataset D is a growing sequence of n-dimensional vectors, i.e., it is of the form $D := \{x_1, x_2, x_3, \ldots, x_t, x_{t+1}, \ldots\}$, with $x_i \in \mathbb{R}^n$, for $i \geq 1$.*

For multiple streams, we assume all measurements are available together at a single site (i.e., the vector consisting of all measurements for a single time tick is available at some node). Distributed processing is beyond the scope of this thesis. Still, the continuous arrival of data as well as the large number of streams pose significant challenges for incremental, any-time mining.

1.5.1 Example domains and applications

In this section we describe briefly some application domains where time series stream data arise and give some illustrative, intuitive examples.

Sensor data Sensors are small devices that gather measurements of physical systems, both natural and artificial.

- Examples of the former are sensors for temperature, humidity, light intensity (e.g., from the sun) and other environmental parameters, river gauges, geological and seismological observations, patient physiological data, etc.

- Sensors are also being embedded to monitor human-made structures, such as measuring bridge or building vibrations, road traffic, chlorine concentration within drinkable water distribution networks, etc.

There are numerous, fascinating applications for such sensors and sensor networks, in fields such as health care and monitoring, industrial process control, civil infrastructure, road traffic safety and smart houses, to mention a few. Some examples of interesting patterns would be:

- Automobile traffic follows a clear daily periodicity. Also, in each day there is another distinct pattern (morning and afternoon rush hours). However, at an hour scale traffic is highly irregular and bursty.
- All sensors measuring chlorine concentration follow the same periodic trend, which a human may map to the normal water demand pattern. However, at some point in time, a new, uncorrelated trend may appear, e.g., due to a leak affecting water flow in the neighbourhood of some sensors or due to contamination of the water distribution network.

Network monitoring Detailed network usage and profiling data are collected in modern telecommunications networks.

- One of the main goals are performance monitoring and analysis, which in turn will help in improved network planning (routing, resource allocation and provisioning, etc) [GKMS01, LPC+04].
- Furthermore, numerous network probes are deployed today in the Internet, to aid in anomaly and intrusion detection. Such data are collected from multiple locations into large repositories [ISC] and subsequently analysed.

Some examples of interesting patterns may be:

- Network traffic follows a periodic pattern, with occasional spikes during Christmas. This may help in provisioning network resources as necessary.
- Traffic flows exhibit high correlations and the set of traffic flows may be summarised by a few hidden variables, each describing a correlated trend. Which nodes participate in each trend may provide useful insights for network planning.
- There is a sudden increase in traffic to a set of machines. Furthermore, this new traffic is highly correlated among these machines. This may be an indication of a distributed denial of service (DDOS) attack, which may be hard to discover by simpler methods (such per-server traffic rate monitoring and naive thresholding).

Financial applications In the financial domain there are several sources of time series stream data.

- A typical example is stock quotes, which generate thousands of streams updated every few minutes [ZS02]. The analysis of market trends and correlations is a very important application.
- Furthermore, financial institutions collect and maintain account historical data, such as balance or transfer activity over time. Again, trends and correlations in such time series may help in planning as well as fraud detection.

In all of the above cases, incremental processing of such data and any-time reporting of trends and correlations is an important but difficult task. Some possible patterns of interest would be:

- The market generally follows, e.g., daily and yearly cycles. At the current moment, the overall trend is falling and most stocks reflect that. However, a group of, e.g., some bio-tech companies exhibit high correlation among themselves and do *not* follow the global trend.
- Account balances follow a certain trend. However, for the past couple of days, there are suspicious correlations among some of these accounts.

1.6 Thesis overview

Our main technique is *multi-resolution analysis*. We also use ideas from signal processing, information theory and compression. The outline of the thesis is shown in Figure 1.1. Next, we describe each of the parts in more detail.

1.6.1 Part I — Homogeneous spatial data

In this part we consider homogeneous spatial data, where each point has only spatial features. More precisely, all data points belong to a multidimensional vector space (see Definition 4) or, more generally, to a metric space[1]. Our techniques rely on examination of the dataset at multiple distance scales. We employ both the first and the second moments of pairwise distance distributions. More specifically, our main tools are the *local correlation integral* (see

[1] A *metric space* is a set of points together with a distance function that satisfies the triangle inequality.

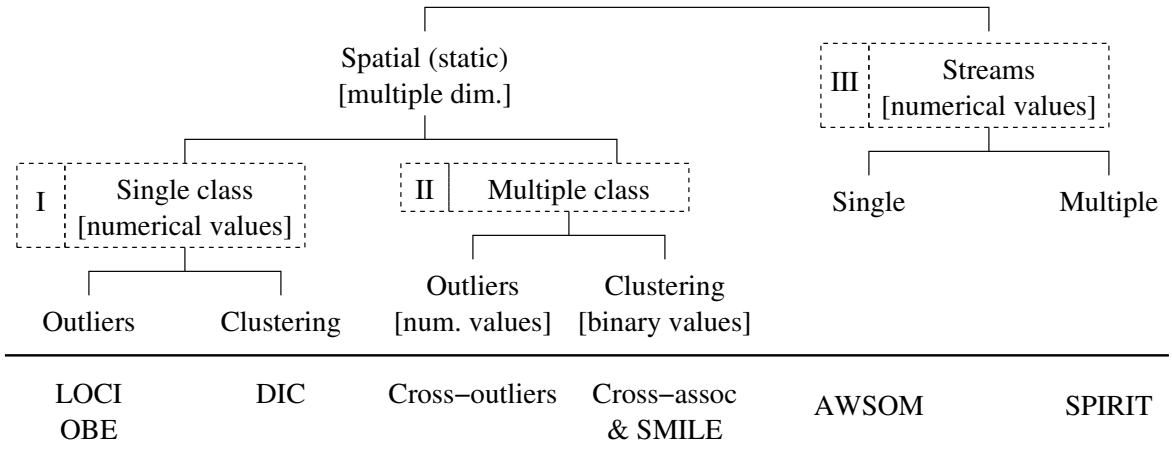


Figure 1.1: Thesis outline.

Chapter 3), which is essentially the cumulative distribution function (CDF) of pairwise distances restricted to a local neighbourhood around each point in the dataset, as well as *neighbour growth curves* (see Chapter 5), which describe how the number of neighbours around each point grows with respect to distance.

In Chapter 3 we we propose a new method for evaluating "outlier-ness," which we call the *Local Correlation Integral* (*LOCI*) [PKGF03]. As with the best previous methods, LOCI is highly effective for detecting outliers and groups of outliers (also known as micro-clusters). In addition, it offers the following advantages and novelties: *(a)* It provides an automatic, data-dictated cut-off to determine whether a point is an outlier—in contrast, previous methods force users to pick cut-offs, without any hints as to what cut-off value is best for a given dataset. *(b)* It can provide a LOCI plot for each point; this plot summarises a wealth of information about the data in the vicinity of the point, determining clusters, micro-clusters, their diameters and their inter-cluster distances. None of the existing outlier-detection methods can match this feature, because they output only a single number for each point: its outlierness score. *(c)* Our LOCI method can be computed as quickly as the best previous methods. *(d)* Moreover, LOCI leads to a practically linear approximate method, *aLOCI* (for *approximate LOCI*), which provides fast highly-accurate outlier detection. To the best of our knowledge, this is the first work to use approximate computations to speed up outlier detection.

In Chapter 4 we present *outliers by example (OBE)* [ZKPF04] to incorporate user feedback, without exposing users to any parameters. Instead, we try to infer the (few) parameters from user examples. A fundamental issue is that the notion of which objects are outliers sometimes varies between users or, even, datasets. We present a novel, user-friendly solution to this problem, by bringing users into the loop. Our OBE (Outlier By Example) system

is, to the best of our knowledge, the first that allows users to give some examples of what they consider as outliers. Then, it can directly incorporate a small number of such examples to successfully discover the hidden concept and spot further objects that exhibit the same "outlier-ness" as the examples.

In Chapter 5 we present *dimension induced clustering (DIC)* [GHPT05] and develop a parameter-free transformation of high-dimensional points into a pair of local dimension and local density features, which can be used to cluster points belonging to low-dimensional manifolds. It is commonly assumed that high-dimensional datasets contain points most of which are located in low-dimensional manifolds. Detection of low-dimensional clusters is an extremely useful task for performing operations such as clustering and classification, nevertheless, it is a very challenging computational problem. We study the problem of finding subsets of points with low intrinsic dimensionality. Our main contribution is to extend the definition of fractal correlation dimension, which measures average volume growth rate, in order to estimate the intrinsic dimensionality of the data in local neighbourhoods. We provide a careful analysis of several key examples in order to demonstrate the properties of our measure. Based on our proposed measure, we introduce a novel approach to discover clusters with low dimensionality. The resulting algorithms extend previous density based measures, which have been successfully used for clustering.

1.6.2 Part II — Heterogeneous spatial data

In the first part, we consider "pure" spatial data. In this part we develop mining methods when each point has one or more binary attributes (or features) associated with it, besides its spatial location (see Definition 5. For example, galaxies may belong to one of two types (say, spiral or elliptical) or patches of land may contain several among tens or hundreds of different species.

In Chapter 8 we introduce cross-outliers [PF03]. To the best of our knowledge, work on outliers up to date focuses exclusively on the problem as follows [Haw80]: "given a *single* set of observations in some space, find those that deviate so as to arouse suspicion that they were generated by a different mechanism." However, in several domains, we have more than one set of observations (or, equivalently, as single set with class labels assigned to each observation). A single observation may look normal both within its own class, as well as within the entire set of observations. However, when examined with respect to other classes, it may still arouse suspicions. Thus, we consider the problem "given a set of observations with class labels, find those that arouse suspicions, taking into account the

class labels." Many of the existing outlier detection approaches cannot be extended to this case. We present one practical approach for dealing with this problem.

In Chapter 9 we consider spatial data consisting of a set of *binary features* taking values over a collection of *spatial extents* (grid cells) and we propose a method that simultaneously finds spatial correlation and feature co-occurrence patterns, without *any* parameters [PGT+05]. In particular, we employ the Minimum Description Length (MDL) principle coupled with a natural way of compressing regions. This defines what "good" means: a feature co-occurrence pattern is good, if it helps us better compress the set of locations for these features. Conversely, a spatial correlation is good, if it helps us better compress the set of features in the corresponding region. Our approach is scalable for large datasets (both number of locations and of features).

1.6.3 Part III — Streams

In this part, we consider numerical, time series streams (see Definition 1.5) and develop methods to capture trends at multiple time scales on a single stream, as well as correlations among multiple streams.

In Chapter 12 we present *AWSOM (Arbitrary Window Stream mOdeling Method)* [PBF03] which allows us to make long range forecasts using limited resources. It allows us to efficiently and effectively discover interesting patterns and trends. This can be done automatically, i.e., with no prior inspection of the data or any user intervention and expert tuning before or during data gathering. Our algorithms require limited resources and can thus be incorporated in sensors—possibly alongside a distributed query processing engine [CCC+02, BGS01, MSHR02]. Updates are performed in constant time with respect to stream size, using logarithmic space. Existing forecasting methods (SARIMA, GARCH, etc) or "traditional" Fourier and wavelet analysis fall short on one or more of these requirements. To the best of our knowledge, AWSOM is the first framework that combines all of the above characteristics.

In Chapter 13 we present *SPIRIT (Streaming Pattern dIscoveRy in multIple Time-series)* [PSF05]. Given n numerical data streams, all of whose values we observe at each time tick t, SPIRIT can incrementally find correlations and hidden variables, which summarise the key trends in the entire stream collection. It can do this quickly, with no buffering of stream values and without comparing pairs of streams. Moreover, it is any-time, single pass, and it dynamically detects changes. The discovered trends can also be used to immediately spot potential anomalies, to do efficient forecasting and, more generally, to dramatically simplify further

data processing.

Part I

Spatial mining — Homogeneous

Chapter 2

Introduction

In this part we consider homogeneous spatial data, where each point has only spatial features. More precisely, all data points belong to a multidimensional vector space (see Definition 4). More generally, they may belong to a metric space, where only a pairwise distance function that satisfies the triangle inequality is required. Spatial data arise in several applications, such as biodiversity, geographical and astrophysics data, as well as multimedia and information retrieval, to mention a few (see also Section 1.4.1).

Our techniques rely on examination of the dataset at multiple distance scales. We employ both the first and the second moments of pairwise distance distributions. More specifically, our main tools are the *local correlation integral* (see Chapter 3), which is essentially the cumulative distribution function (CDF) of pairwise distances restricted to a local neighbourhood around each point in the dataset, as well as *neighbour growth curves* (see Chapter 5), which describe how the number of neighbours around each point grows with respect to distance.

2.1 Outlier detection

In many applications (e.g., fraud detection, financial analysis and health monitoring), rare events and exceptions among large collections of objects are often more interesting than the common cases. Consequently, there is increasing attention on methods for discovering such "exceptional" objects in large datasets and several approaches have been proposed [AY01, AAR96, BL94, BKNS00, JKM99, JKN98, KN97, KN98, KN99, KNT00].

In Chapter 3 we lay the foundations for LOCI, a novel outlier detection method which examines the dataset at multiple distance scales. As with the best previous methods, LOCI is highly effective for detecting outliers and groups of outliers (also known as micro-clusters).

In addition, it provides a data-dictated cut-off to determine whether a point is an outlier—in contrast, previous methods force users to pick cut-offs, without any hints as to what cut-off value is best for a given dataset. LOCI can also provide a LOCI plot for each point; this plot summarises a wealth of information about the data in the vicinity of the point, determining clusters, micro-clusters, their diameters and their inter-cluster distances. None of the existing outlier-detection methods can match this feature, because they output only a single number for each point: its outlier-ness score In Chapter 4 we present OBE (Outlier By Example), a method to incorporate user feedback, without exposing the users to *any* parameters. Instead, we infer those parameters automatically, from a list of examples given by the user. Our OBE system is, to the best of our knowledge, the first that allows users to give some examples of what they consider as outliers. Then, it can directly incorporate a small number of such examples to successfully discover the hidden concept and spot further objects that exhibit the same "outlier-ness" as the examples.

2.2 Dimension induced clustering

Real datasets exhibit patterns and regularities. As a main consequence, points typically lie on low-dimensional manifolds, rather than being evenly spread out. Detecting subsets of points with low intrinsic dimensionality is useful in tasks such as indexing and classification. Furthermore, separating points based on some notion of "local dimensionality" is helpful in identifying subsets of points that are qualitatively different.

In Chapter 5 we draw upon ideas similar to those used in LOCI to intuitively define a topological notion of dimension, which does not depend on the notion of a linear subspace. Furthermore, our algorithms for identifying low-dimensional manifolds are not sensitive to the dimension of the original space and, thus, do not suffer from the "curse of dimensionality". We develop a parameter-free transformation of high-dimensional points (or, in general, points in a metric space) into a pair of local dimension and local density features, which can be used to cluster points belonging to low-dimensional manifolds. Our resulting algorithms extend previous density based measures, which have been successfully used for clustering.

Chapter 3

Outlier detection

In this chapter we propose a new method (LOCI—LOcal Correlation Integral method) for finding outliers in large, multidimensional data sets. The main contributions of our work can be summarised as follows:

- We introduce the *multi-granularity deviation factor* (MDEF), which can cope with local density variations in the feature space and detect both isolated outliers as well as outlying clusters. Our definition is simpler and more intuitive than previous attempts to capture similar concepts [BKNS00]. This is important, because the users who interpret the findings of an outlier detection tool and make decisions based on them are likely to be domain experts, not KDD experts.
- We propose a novel (statistically intuitive) method that selects a point as an outlier if its MDEF value deviates significantly (more than three standard deviations) from the local averages. We also show how to quickly estimate the average and standard deviation of MDEF values in a neighbourhood. Our method is particularly appealing, because it provides an automatic, data-dictated cut-off for determining outliers, by taking into account the distribution of distances between pairs of objects.
- We present several outlier detection schemes and algorithms using MDEF. Our LOCI algorithm, using an exact computation of MDEF values, is at least as fast as the best previous methods.
- We show how MDEF lends itself to a much faster, approximate algorithm (aLOCI) that still yields high-quality results. In particular, because the MDEF is associated with the *correlation integral* [BF95, TTPF01], it is an aggregate measure. We show how approximation methods such as *box counting* can be used to reduce the computational cost to only $O(kN)$, i.e., linear both with respect to the data set size N and the number

of dimensions k. Previous methods are considerably slower, because for each point, they must iterate over every member of a local neighbourhood or cluster; aLOCI does not.

- We extend the usual notion of an "outlier-ness" score to a more informative *LOCI plot*. Our method computes a LOCI plot for each point; this plot summarises a wealth of information about the points in its vicinity, determining clusters, micro-clusters, their diameters and their inter-cluster distances. Such plots can be displayed to the user, as desired. For example, returning the LOCI plots for the set of detected outliers enables users to drill down on outlier points for further understanding. None of the existing outlier-detection methods can match this feature, because they restrict themselves to a single number as an outlier-ness score.

- We present extensive experimental results using both real world and synthetic data sets to verify the effectiveness of the LOCI method. We show that, in practice, the algorithm scales linearly with data size and with dimensionality. We demonstrate the time-quality trade-off by comparing results from the exact and approximate algorithms. The approximate algorithm can, in most cases, detect all outstanding outliers very efficiently.

To the best of our knowledge, this is the first work to use approximate computations to speed up outlier detection. Using fast approximate calculations of the aggregates computed by an outlier detection algorithm (such as the number of neighbours within a given distance) makes a lot of sense for large databases. Considerable effort has been invested toward finding good measures of distance. However, very often it is quite difficult, if not impossible, to precisely quantify the notion of "closeness". Furthermore, as the data dimensionality increases, it becomes more difficult to come up with such measures. Thus, there is already an inherent fuzziness in the concept of an outlier and any outlier score is more of an informative indicator than a precise measure.

This chapter is organised as follows. In Section 3.1 we give a brief overview of related work on outlier detection. Section 3.2 introduces the LOCI method and describes some basic observations and properties. Section 3.3 describes our LOCI algorithm, while Section 3.4 describes our aLOCI algorithm. Section 3.5 presents our experimental results, and we conclude in Section 3.6.

Figure 3.1: (a) Local density problem, and (b) multi-granularity problem

3.1 Related work

The existing approaches to outlier detection can be classified into the following five categories.

Distribution-based approach. Methods in this category are typically found in statistics textbooks. They deploy some standard distribution model (e.g., Normal) and flag as outliers those objects which deviate from the model [BL94, Haw80, RL87]. However, most distribution models typically apply *directly* to the feature space and are univariate (i.e., have very few degrees of freedom). Thus, they are unsuitable even for moderately high-dimensional data sets. Furthermore, for arbitrary data sets without any prior knowledge of the distribution of points, we have to perform expensive tests to determine which model fits the data best, if any!

Depth-based approach. This is based on computational geometry and computes different layers of k-d convex hulls [JKN98]. Objects in the outer layer are detected as outliers. However, it is well-known that these algorithms suffer from the dimensionality curse and cannot cope with large k.

Clustering approach. Many clustering algorithms detect outliers as by-products [JMF99]. However, since the main objective is clustering, they are not optimised for outlier detection. Furthermore, in most cases, the outlier detection criteria are implicit and cannot easily be inferred from the clustering procedures. An intriguing clustering algorithm using the fractal dimension has been suggested by [BC00]; however it has not been demonstrated on real

datasets.

The above three approaches for outlier detection are not appropriate for high-dimensional, large, arbitrary data sets. However, this is often the case with KDD in large databases. The following two approaches have been proposed and are attracting more attention.

Distance-based approach. This was originally proposed by E.M. Knorr and R.T. Ng [KN97, KN98, KN99, KNT00, BS03a]. An object in a data set P is a *distance-based outlier* if at least a fraction β of the objects in P are further than r from it. This outlier definition is based on a single, global criterion determined by the parameters r and β. This can lead to problems when the data set has both dense and sparse regions [BKNS00] (see Figure 3.1(a); either the left outlier is missed or every object in the sparse cluster is also flagged as an outlier).

Furthermore, all of the above approaches regard being an outlier as a binary property. They do not take into account both the degree of "outlier-ness" and where the "outlier-ness" is presented.

Density-based approach. This was proposed by M. Breunig, et al. [BKNS00]. It relies on the *local outlier factor* (*LOF*) of each object, which depends on the local density of its neighbourhood. The neighbourhood is defined by the distance to the *MinPts*-th nearest neighbour. In typical use, objects with a high LOF are flagged as outliers. W. Jin, et al. [JTH01] proposed an algorithm to efficiently discover top-n outliers using clusters, for a particular value of *MinPts*.

LOF does not suffer from the local density problem. However, selecting *MinPts* is non-trivial. In order to detect outlying clusters, *MinPts* has to be as large as the size of these clusters (see Figure 3.1(b); if we use a "shortsighted" definition of a neighbourhood—i.e., too few neighbours—then we may miss small outlying clusters), and computation cost is directly related to *MinPts*. Furthermore, the method exhibits some unexpected sensitivity on the choice of *MinPts*. For example, suppose we have only two clusters, one with 20 objects and the other with 21 objects. For $MinPts = 20$, *all* objects in the smaller cluster have large LOF values, and this affects LOF values over any range that includes $MinPts = 20$.

In contrast, LOCI automatically flags outliers, based on probabilistic reasoning. Also, MDEF is not so sensitive to the choice of parameters, as in the above 20-21 clusters example. Finally, LOCI is well-suited for fast, one pass, $O(kN)$ approximate calculation. Although some algorithms exist for approximate nearest neighbour search [AMN$^+$98, Ber93, GIM99], it seems unlikely that these can be used to achieve $O(kN)$ time with LOF. Our method uses

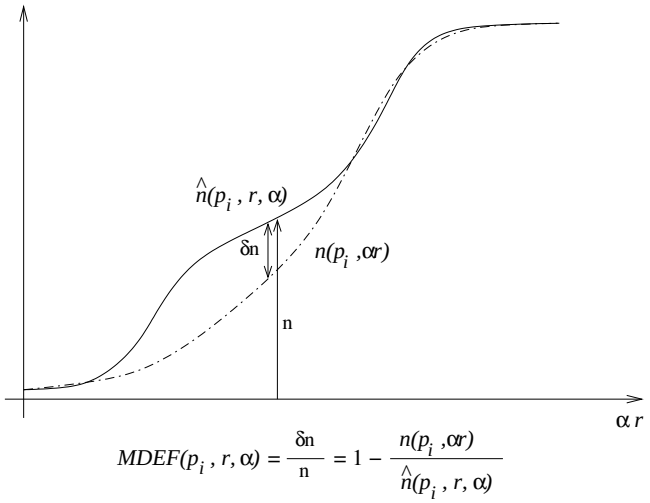


Figure 3.2: Estimation of MDEF from the local correlation integral and neighbour count functions. The dashed curve is the number of αr-neighbours of p_i and the solid curve is the average number of αr-neighbours over the r-neighbourhood (i.e., sampling neighbourhood) of p_i.

an aggregate measure (the proposed local correlation integral) that relies strictly on counts. Because it can be estimated (with box-counting) *without* iterating over every point in a set, it can easily cope with multiple granularities, without an impact on speed.

3.2 Proposed method

One can argue that, intuitively, an object is an "outlier" if it is in some way "significantly different" from its "neighbours." Two basic questions that arise naturally are:

- What constitutes a "neighbourhood?"
- How do we determine "difference" and whether it is "significant?"

Inevitably, we have to make certain choices. Ideally, these should lead to a definition that satisfies the following, partially conflicting criteria:

- It is intuitive and easy to understand: Those who interpret the results are experts in their domain and not on outlier detection.
- It is widely applicable and provides reasonable flexibility: Not everyone has the same idea of what constitutes an outlier and not all data sets conform to the same, specific rules (if any).

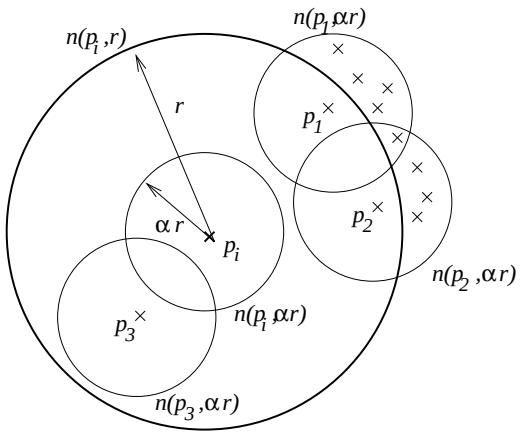


Figure 3.3: Definitions for n and $\hat{n}$—for instance $n(p_i, r) = 4$, $n(p_r, \alpha r) = 1$, $n(p_1, \alpha r) = 6$ and $\hat{n}(p_i, r, \alpha) = (1 + 6 + 5 + 1)/4 = 3.25$.

- It should lend itself to fast computation: This is obviously important with today's ever-growing collections of data.

3.2.1 Multi-granularity deviation factor (MDEF)

In this section, we introduce the multi-granularity deviation factor (MDEF), which satisfies the properties listed above. Let the r-neighbourhood of an object p_i be the set of objects within distance r of p_i.

Intuitively, the MDEF at radius r for a point p_i is the relative deviation of its local neighbourhood density from the average local neighbourhood density in its r-neighbourhood. Thus, an object whose neighbourhood density matches the average local neighbourhood density will have an MDEF of 0. In contrast, outliers will have MDEFs far from 0.

To be more precise, we define the following terms (Table 3.1 describes all symbols and basic definitions). Let $n(p_i, \alpha r)$ be the number of objects in the αr-neighbourhood of p_i. Let $\hat{n}(p_i, r, \alpha)$ be the average, over all objects p in the r-neighbourhood of p_i, of $n(p, \alpha r)$ (see Figure 3.3). The use of two radii serves to decouple the neighbour size radius αr from the radius r over which we are averaging. We denote as the *local correlation integral* the function $\hat{n}(p_i, \alpha, r)$ over all r.

Definition 7 (MDEF). *For any p_i, r and α we define the* multi-granularity deviation factor

Symbol	Definition
$\mathbb{P}$	Set of objects $\mathbb{P} = \{p_1, \ldots, p_i, \ldots, p_N\}$.
p_i	Object in the dataset $\mathbb{P}$, for $1 \leq i \leq N$.
N	Data set size ($\lvert\mathbb{P}\rvert \equiv N$).
k	Dimension of data set, i.e., when $\mathbb{P}$ is a vector space, $p_i = (p_i^1, p_i^2, \ldots, p_i^k)$.
$d(p_i, p_j)$	Distance between p_i and p_j.
$R_{\mathbb{P}}$	Point set radius, i.e., $R_{\mathbb{P}} \equiv \max_{p_i, p_j \in \mathbb{P}} d(p_i, p_j)$.
$NN(p_i, m)$	The m-th nearest neighbour of object p_i ($NN(p_i, 0) \equiv p_i$).
$\mathcal{N}(p_i, r)$	The set of r-neighbours of p_i, i.e., $$\mathcal{N}(p_i, r) \equiv \{p \in \mathbb{P} \mid d(p, p_i) \leq r\}$$ Note that the neighbourhood contain p_i itself, thus the counts can never be zero.
$n(p_i, r)$	The number of r-neighbours of p_i, i.e., $n(p_i, r) \equiv \lvert\mathcal{N}(p_i, r)\rvert$.
$\hat{n}(p_i, r, \alpha)$	Average of $n(p, \alpha r)$ over the set of r-neighbours of p_i, i.e., $$\hat{n}(p_i, r, \alpha) \equiv \frac{\sum_{p \in \mathcal{N}(p_i, r)} n(p, \alpha r)}{n(p_i, r)}$$
$\hat{\sigma}(p_i, r, \alpha)$	Standard deviation of $n(p, \alpha r)$ over the set of r-neighbours, i.e., $$\hat{\sigma}(p_i, r, \alpha) \equiv \sqrt{\frac{\sum_{p \in \mathcal{N}(p_i, r)} (n(p, \alpha r) - \hat{n}(p_i, r, \alpha))^2}{n(p_i, r)}}$$ When clear from the context ($\hat{n}$), we use just $\hat{\sigma}$.
$MDEF(p_i, r, \alpha)$	Multi-granularity deviation factor for point p_i at radius (or scale) r.
$\sigma_{MDEF}(p_i, r, \alpha)$	Normalised deviation (thus, directly comparable to $MDEF$).
k_σ	Determines what is *significant* deviation, i.e., points are flagged as outliers iff $$MDEF(p_i, r, \alpha) > k_\sigma \sigma_{MDEF}(p_i, r, \alpha)$$ We fix this value to $k_\sigma = 3$ (see Lemma 1).
$\mathcal{C}(p_i, r, \alpha)$	Set of cells on some grid, with cell side $2\alpha r$, each fully contained within $\mathcal{L}_\infty$-distance r from object p_i.
C_i	Cell in some grid.
c_i	The object count within the corresponding cell C_i.
$S_q(p_i, r, \alpha)$	Sum of box counts to the q-th power, i.e., $$S_q(p_i, r, \alpha) \equiv \sum_{C_i \in \mathcal{C}(p_i, r, \alpha)} c_i^q$$

Table 3.1: Symbols and definitions.

(MDEF) at radius (or scale) r as:

$$MDEF(p_i, r, \alpha) = \frac{\hat{n}(p_i, r, \alpha) - n(p_i, \alpha r)}{\hat{n}(p_i, \alpha, r)} \tag{3.1}$$

$$= 1 - \frac{n(p_i, \alpha r)}{\hat{n}(p_i, \alpha, r)} \tag{3.2}$$

See Figure 3.2. Note that the r-neighbourhood for an object p_i always contains p_i. This implies that $\hat{n}(p_i, \alpha, r) > 0$ and so the above quantity is always defined.

For faster computation of MDEF, we will sometimes *estimate* both $n(p_i, \alpha r)$ and $\hat{n}(p_i, r, \alpha)$. This leads to the following definitions:

Definition 8 (Counting and sampling neighborhood). *The* counting neighbourhood *(or αr-neighbourhood) is the neighbourhood of radius αr, over which each $n(p, \alpha r)$ is estimated. The* sampling neighbourhood *(or r-neighbourhood) is the neighbourhood of radius r, over which we collect samples of $n(p, \alpha r)$ in order to estimate $\hat{n}(p_i, r, \alpha)$.*

In Figure 3.3, for example, the large circle bounds the sampling neighbourhood for p_i, while the smaller circles bound counting neighbourhoods for various p (see also Figure 3.2).

The main outlier detection scheme we propose relies on the standard deviation of the αr-neighbour count over the sampling neighbourhood of p_i. We thus define the following quantity

$$\sigma_{MDEF}(p_i, r, \alpha) = \frac{\hat{\sigma}(p_i, r, \alpha)}{\hat{n}(p_i, r, \alpha)} \tag{3.3}$$

which is the normalised standard deviation $\hat{\sigma}(p_i, r, \alpha)$ of $n(p, \alpha r)$ for $p \in \mathcal{N}(p_i, r)$ (in Section 3.4 we present a fast, approximate algorithm for estimating σ_{MDEF}).

The main reason we use an *extended* neighbourhood ($\alpha < 1$) for sampling is to enable fast, approximate computation of MDEF as explained in Section 3.4. Besides this, $\alpha < 1$ is desirable in its own right to deal with certain singularities in the object distribution (we do not discuss this due to space considerations).

Advantages of our definitions. Among several alternatives for an outlier score (such as $\max(\hat{n}/n, n/\hat{n})$, to give one example), our choice allows us to use probabilistic arguments for flagging outliers. This is a very important point and is exemplified by Lemma 1 in Section 3.2.2. The above definitions and concepts make minimal assumptions. The only general requirement is that a distance is defined. Arbitrary distance functions are allowed,

which may incorporate domain-specific, expert knowledge, if desired. Furthermore, the standard deviation scheme assumes that pairwise distances *at a sufficiently small scale* are drawn from a single distribution, which is reasonable.

For the fast approximation algorithms, we make the following additional assumptions (the exact algorithms do not depend on these):

- Objects belong to a k-dimensional vector space, i.e., $p_i = (p_i^1, p_i^2, \ldots, p_i^k)$. This assumption holds in most situations. However, if the objects belong to an arbitrary metric space, then it is possible to embed them into a vector space. There are several techniques for this [CNBYM01] which use the $\mathcal{L}_\infty$ norm on the embedding vector space[1].

- We use the $\mathcal{L}_\infty$ norm, which is defined as $\|p_i - p_j\|_\infty \equiv \max_{1 \le m \le k} |p_i^m - p_j^m|$. This is not a restrictive hypothesis, since it is well-known that, in practice, there are no clear advantages of one particular norm over another [FLM77, GIM99].

3.2.2 LOCI outlier detection

In this section, we describe and justify our main outlier detection scheme. It should be noted that, among all alternatives in the problem space LOCI can be easily adapted to match several choices. It computes the necessary summaries in one pass and the rest is a matter of interpretation.

In particular, given the above definition of MDEF, we still have to make a number of decisions. In particular, we need to answer the following questions:

- **Sampling neighbourhood:** Which points constitute the sampling neighbourhood of p_i, or, in other words, which points do we average over to compute $\hat{n}$ (and, in turn, MDEF) for a p_i in question?

- **Scale:** Regardless of the choice of neighbourhood, over what range of distances do we compare n and $\hat{n}$?

- **Flagging:** After computing the MDEF values (over a certain range of distances), how do we use them to choose the outliers?

[1] Given objects π_i in a metric space $\mathbb{M}$ with distance function $\delta(\pi_i, \pi_j)$, one typical approach is to choose k landmarks $\{\Pi_1, \ldots, \Pi_k\} \subseteq \mathbb{M}$ and map each object π_i to a vector with components $p_i^j = \delta(\pi_i, \Pi_j)$.

LOCI outlier detection method. The proposed LOCI outlier detection method answers the above questions as follows. Advantages and features of LOCI are due to these design choices combined with inherent properties of MDEF.

- **Large sampling neighbourhood:** For each point and counting radius, the sampling neighbourhood is selected to be large enough to contain enough samples. We choose $\alpha = 1/2$ in all exact computations, and we typically use $\alpha = 1/16$ in aLOCI (introduced in Section 3.4) for robustness (particularly in the estimation of σ_{MDEF}).

- **Full-scale:** The MDEF values are examined for a wide range of sampling radii. In other word, the maximum sampling radius is $r_{max} \approx \alpha^{-1} R_{\mathbb{P}}$ (which corresponds to maximum counting radius of $R_{\mathbb{P}}$). The minimum sampling radius r_{min} is determined based on the number of objects in the sampling neighbourhood. We always use a smallest sampling neighbourhood with $\hat{n}_{min} = 20$ neighbours; in practice, this is small enough but not too small to introduce statistical errors in MDEF and σ_{MDEF} values.

- **Standard deviation-based flagging**: A point is flagged as an outlier, if for *any* $r \in [r_{min}, r_{max}]$ its MDEF is *sufficiently* large, i.e.,

$$MDEF(p_i, r, \alpha) > k_\sigma \sigma_{MDEF}(p_i, r, \alpha)$$

  In all our experiments, we use $k_\sigma = 3$ (see Lemma 1).

The standard deviation-based flagging is one of the main features of the LOCI method. It replaces any "magic cut-offs" with probabilistic reasoning based on σ_{MDEF}. It takes into account *distribution* of *pairwise distances* and compares each object to those in its sampling neighbourhood. Note that, even if the global distribution of distances varies significantly (e.g., because it is a mixture of very different distributions), the use of the *local* deviation successfully solves this problem. In fact, in many *real* data sets, the distribution of pairwise distances follows a specific distribution over all or most scales [TTPF01, BF95]. Thus, this approach works well for many real data sets. The user may alter the minimum neighbourhood size r_{min} and k_σ if so desired, but in practice this is unnecessary.

Lemma 1 (Deviation probability bounds). *For any distribution of pairwise distances, and for any randomly selected p_i, we have*

$$Pr\{MDEF(p_i, r, \alpha) > k_\sigma \sigma_{MDEF}(p_i, r, \alpha)\} \leq \frac{1}{k_\sigma^2}$$

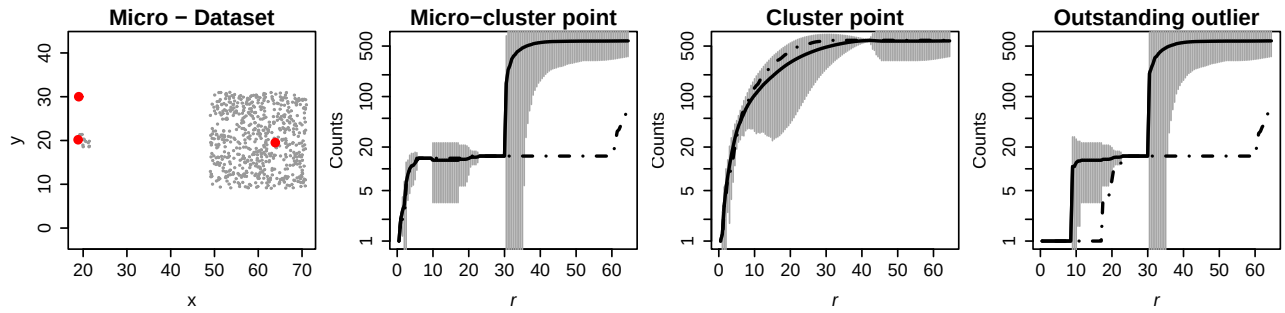


Figure 3.4: LOCI plots from an actual dataset—see also Section 3.5.

Proof. From Chebyshev's inequality it follows that, for a given point p_i and radius r,

$$\begin{aligned} &\Pr\{MDEF(p_i, r, \alpha) > k_\sigma \sigma_{MDEF}(p_i, r, \alpha)\} \\ &\leq \Pr\{|MDEF(p_i, r, \alpha)| > k_\sigma \sigma_{MDEF}(p_i, r, \alpha)\} \\ &\leq \sigma^2_{MDEF}(p_i, r, \alpha) / (k_\sigma \sigma_{MDEF}(p_i, r, \alpha))^2 = 1/k^2_\sigma \quad . \end{aligned}$$

□

This is a relatively loose bound, but it holds regardless of the distribution. For known distributions, the actual bounds are tighter; for instance, if the neighbourhood sizes follow a normal distribution and $k_\sigma = 3$, much less than 1% of the points should deviate by that much (as opposed to $\approx 10\%$ suggested by the above bound).

3.2.3 Alternative interpretations

As mentioned in Section 3.2.2, we have a range of design choices for outlier detection schemes. Different answers give rise to different outlier detection schemes and provide the user with alternative views. We should emphasise that, if the user want, LOCI can be adapted to *any* desirable interpretation, without any re-computation. Our fast algorithms estimate all the necessary quantities with a single pass over the data and build the appropriate "summaries," no matter how they are later interpreted.

Sampling neighbourhood: Small vs. large. The choice depends on whether we are interested in the deviation of p_i from a small (highly local) or a relatively large neighbourhood. Since LOCI employs standard deviation-based flagging, a sampling neighbourhood large enough to get a sufficiently large sample is desirable. However, when the distance distribution varies widely (which rarely happens, except at *very* large radii) or if the user chooses

non-deviation based scheme (which, although possible, is not recommended) this is an option.

Scale: Single vs. range and distance-based vs. population-based. Regardless of sampling neighbourhood, users could choose to examine MDEF and σ_{MDEF} at either a single radius (which is very close to the distance-based approach [KN99]) or a limited range of radii (same for all the points). Alternatively, they may implicitly specify the radius (or radii) by neighbourhood size (effectively varying the radius at each p_i, depending on density). Either approach might make sense.

Flagging: Thresholding vs. ranking vs. standard deviation-based. Use of the standard deviation is our main contribution and the recommended approach. However, we can easily match previous methods either by "hard thresholding" (*if* we have prior knowledge about what to expect of distances and densities) or "ranking" (if we want to catch a few "suspects" blindly and, probably, "interrogate" them manually later).

3.2.4 LOCI plot

In this section we introduce the *LOCI plot*. This is a powerful tool, no matter what outlier detection scheme is employed. It can be constructed instantly from the computed "summaries" for any point p_i the user desires and it gives a wealth of information about the vicinity of p_i: why it is an outlier with regard to its vicinity, as well as information about nearby clusters and micro-clusters, their diameters and inter-cluster distances.

Definition 9 (LOCI plot). *For any object p_i, the plot of $n(p_i, \alpha r)$ and $\hat{n}(p_i, r, \alpha)$ with $\hat{n}(p_i, r, \alpha) \pm 3\hat{\sigma}(p_i, r, \alpha)$, versus r (for a range of radii of interest), is called its* LOCI plot.

We give detailed examples from actual datasets in Section 3.5. Here we briefly introduce the main features (see also Figure 3.4). The solid line shows $\hat{n}$ and the dashed line is n is all plots.

- Consider the point in the micro-cluster (at $x = 18, y = 20$). The n value looks similar up to the distance (roughly 30) we encounter the large cluster. Earlier, the increase in deviation (in the range of $\approx$ 10–20) indicates the presence of a (small) cluster. Half the width (since $\alpha = 1/2$, and the deviation here is affected by the counting radius) of this range (about $10/2 = 5$) is the radius of this cluster.

```
// Pre-processing
Foreach p_i ∈ ℙ:
    Perform a range-search
      for N_i = {p ∈ ℙ | d(p_i, p) ≤ r_max}
    From N_i, construct a sorted list D_i
      of the critical and α-critical distances of p_i
// Post-processing
Foreach p_i ∈ ℙ:
    For each radii r ∈ D_i (ascending):
      Update n(p_i, αr) and n̂(p_i, r, α)
      From n and n̂, compute
        MDEF(p_i, r, α) and σ_MDEF(p_i, r, α)
      If MDEF(p_i, r, α) > 3σ_MDEF(p_i, r, α),
        flag p_i
```

Figure 3.5: The exact LOCI algorithm.

- A similar increase in deviation happens at radius 30, along with an increase in $\hat{n}$. Also, note that n shows a similar jump at $\alpha^{-1} \times 30 = 60$ (this time it is the sampling radius that matters). Thus, ≈ 30 is the distance to the next (larger) cluster.
- In the cluster point (at $x = 64, y = 19$) we see from the middle LOCI plot that the two counts ($\hat{n}$ and $\hat{\sigma}$) are similar, as expected. The increase in deviation, however, provides the information described above for the first increase (here the counting radius matters again, so we should multiply the distances by α).
- The general magnitude of the deviation always indicates how "fuzzy" (i.e., spread-out and inconsistent) a cluster is.
- For the outstanding outlier point (at $x = 18, y = 30$), we see the deviation increase along with the pair of jumps in $\hat{n}$ and n (the distance between the jumps determined by α) *twice*, as we would expect: the first time when we encounter the micro-cluster and the second time when we encounter the large cluster.

3.3 The LOCI algorithm

In this section, we describe our algorithm for detecting outliers using our LOCI method. This algorithm computes exact MDEF and σ_{MDEF} values for all objects, and then reports an outlier whenever MDEF is more than three times larger than σ_{MDEF} for the same radius. Thus the key to a fast algorithm is an efficient computation of MDEF and σ_{MDEF} values.

We can considerably reduce the computation time for MDEF and σ_{MDEF} values by exploiting the following properties:

Observation 1. *For each object p_i and each α, $n(p_i, r)$, $\hat{n}(p_i, r, \alpha)$, and thus $MDEF(p_i, r, \alpha)$ and $\sigma_{MDEF}(p_i, r, \alpha)$ are all piecewise constant functions of r. In particular, $n(p_i, r)$ and $n(p, \alpha r)$ for all p in the r-neighbourhood of p_i can change only when the increase of r causes a new point to be added to either the r-neighbourhood of p_i or the αr-neighbourhood of any of the p.*

This leads to the following definition, where N is the number of objects and $NN(p_i, m)$ is the m-th nearest neighbour of p_i.

Definition 10 (Critical Distance). *For $1 \leq m \leq N$, we call $d(NN(p_i, m), p_i)$ a* critical distance *of p_i and $d(NN(p_i, m), p_i)/\alpha$ an α-*critical distance *of p_i.*

By observation 1, we need only consider radii that are critical or α-critical. Figure 3.5 shows our LOCI algorithm. In a pre-processing pass, we determine the critical and α-critical distances D_i for each object p_i. Then considering each object p_i in turn, and considering increasing radius r from D_i, we maintain $n(p_i, \alpha r)$, $\hat{n}(p_i, r, \alpha)$, $MDEF(p_i, r, \alpha)$, and $\sigma_{MDEF}(p_i, r, \alpha)$. We flag p_i as an outlier if $MDEF(p_i, r, \alpha) > 3\sigma_{MDEF}(p_i, r, \alpha)$ for some r.

The worst-case complexity of this algorithm is $O(N \times (time\_of\_rmax\_range\_search + n_{ub}^2))$, where $n_{ub} = \max\{n(p_i, r_{max}) \mid p_i \in \mathbb{P}\}$. Alternatively, if we specify the range of scales indirectly by numbers of neighbours n_{min} and n_{max} instead of explicit r_{min} and r_{max}, then $r_{min} = d(NN(p_i, n_{min}), p_i)$ and $r_{max} = d(NN(p_i, n_{max}), p_i)$. The complexity of this alternative is $O(N \times (time\_of\_Rmax\_range\_search + n_{max}^2)$, where $R_{max} = \max\{d(NN(p_i, n_{max}), p_i) \mid p_i \in \mathbb{P}\}$. Thus, the complexity of our LOCI algorithm is roughly comparable to that of the best previous density-based approach [BKNS00].

3.4 The aLOCI algorithm

In this section we present our fast, approximate LOCI algorithm (aLOCI). Although algorithms exist for approximate range queries and nearest neighbour search [AMN+98, Ber93, GIM99], applying them directly to previous outlier detection algorithms (or the LOCI algorithm; see Figure 3.5) would not eliminate the high cost of iterating over each object in the (sampling) neighbourhood of each p_i. Yet with previous approaches, *failing* to iterate over each such object means the approach cannot effectively overcome the multi-granularity problem (Figure 3.1(b)). In contrast, our MDEF-based approach is well-suited to fast approximations that avoid these costly iterations, yet are able to overcome the multi-granularity problem. This is because our approach essentially requires only counts at various scales.

// Initialisation
Select set of shifts $S = \{s_0, s_1, \ldots, s_g\}$, where $s_0 = 0$
$l_\alpha = -\lg(\alpha)$
Foreach $s_i \in S$:
  Initialise quadtree $Q(s_i)$
// Pre-processing stage
Foreach $p_i \in \mathbb{P}$:
  Foreach $s_i \in S$:
    Insert p_i in $Q(s_i)$
// Post-processing stage
Foreach $p_i \in \mathbb{P}$:
  Foreach level l:
    Select cell C_i in $Q(s_a)$ with side
      $d_i = R_{\mathbb{P}}/2^l$ and centre closest to p_i
    Select cell C_j in $Q(s_b)$ with side
      $d_j = R_{\mathbb{P}}/2^{l-l_\alpha}$ and centre closest to centre of C_i
    Estimate $MDEF(p_i, \frac{d_j}{2}, \alpha)$ and $\sigma_{MDEF}(p_i, \frac{d_j}{2}, \alpha)$
    If $MDEF(p_i, \frac{d_j}{2}, \alpha) > 3\sigma_{MDEF}(p_i, \frac{d_j}{2}, \alpha)$, flag p_i

Figure 3.6: The approximate aLOCI algorithm.

3.4.1 Definitions and observations

Our aLOCI algorithm is based on a series of observations and techniques outlined in this section.

To quickly estimate the average number of αr-neighbours over all points in an r-neighbourhood of an object $p_i \in \mathbb{P}$ (from now on, we assume $\mathcal{L}_\infty$ distances), we can use the following approach. Consider a grid of cells with side $2\alpha r$ over the set $\mathbb{P}$. Perform a *box count* of the grid: For each cell C_j in the grid, compute the count, c_j, of the number of objects in the cell. Each object in C_j has c_j neighbours in the cell (counting itself), so the total number of neighbours over all objects in C_j is c_j^2. Denote by $\mathcal{C}(p_i, r, \alpha)$ the set of all cells in the grid such that the entire cell is within distance r of p_i. We use $\mathcal{C}(p_i, r, \alpha)$ as an approximation for the r-neighbourhood of p_i. Summing over the entire r-neighbourhood, we get $S_2(p_i, r, \alpha)$, where $S_q(p_i, r, \alpha) \equiv \sum_{C_j \in \mathcal{C}(p_i, r, \alpha)} c_j^q$. The total number of objects is simply the sum of all box counts, i.e., $S_1(p_i, r, \alpha)$.

Lemma 2 (Approximate average neighbor count). *Let $\alpha = 2^{-l}$ for some positive integer l. The average neighbour count over p_i's sampling neighbourhood is approximately:*

$$\hat{n}(p_i, r, \alpha) = \frac{S_2(p_i, r, \alpha)}{S_1(p_i, r, \alpha)}$$

Proof. Follows from the above observations; for details, see [Sch88]. □

However, we need to obtain information at several scales. We can efficiently store cell counts in a k-dimensional quad-tree: The first grid consists of a single cell, namely the bounding box of $\mathbb{P}$. We then recursively subdivide each cell of side $2\alpha r$ into 2^k sub-cells, each with radius αr, until we reach the scale we desire (specified either in terms of its side length or cell count). We keep only pointers to the non-empty child sub-cells in a hash table (typically, for large dimensions k, most of the 2^k children are empty, so this saves considerable space over using an array). For our purposes, we only need to store the c_j values (one number per non-empty cell), and not the objects themselves.

The recursive subdivision of cells dictates the choice[2] of $\alpha = 2^{-l}$ for some positive integer l, since we essentially discretise the range of radii at powers of two.

In addition to approximating $\hat{n}$, our method requires an estimation of $\hat{\sigma}$. The key to our fast approximation of $\hat{\sigma}$ is captured in the following lemma:

Lemma 3 (Approximate std. deviation of neighbor count). *Let $\alpha = 2^{-l}$ for some positive integer l. The standard deviation of the neighbour count is approximately:*

$$\hat{\sigma}(p_i, r, \alpha) = \sqrt{\frac{S_3(p_i, r, \alpha)}{S_1(p_i, r, \alpha)} - \frac{S_2^2(p_i, r, \alpha)}{S_1^2(p_i, r, \alpha)}}$$

Proof. Following the same reasoning as in Lemma 2, the deviation for each object within each cell C_j is $c_j - \hat{n}(p_i, r, \alpha) \approx c_j - S_2(p_i, r, \alpha)/S_1(p_i, r, \alpha)$. Thus, the sum of squared differences for all objects within the cell is $c_j \left(c_j - S_2(p_i, r, \alpha)/S_1(p_i, r, \alpha)\right)^2$. Summing over all cells and dividing by the count of objects $S_1(p_i, r, \alpha)$ gives $\frac{1}{S_1} \sum_j \left(c_j^3 - \frac{2c_j^2 S_2}{S_1} + \frac{c_j S_2^2}{S_1^2} \right) = \frac{S_3}{S_1} - \frac{2S_2^2}{S_1^2} + \frac{S_2^2}{S_1^2}$, which leads to the above result. □

From the above discussion, we see that box counting within quad trees can be used to quickly estimate the MDEF values and σ_{MDEF} values needed for our LOCI approach. However, in practice, there are several important issues that need to be resolved to achieve accurate results, which we address next.

[2] In principle, we can choose any integer power $\alpha = c^{-l}$ by subdividing each cell into c^k sub-cells. However, this makes no difference in practice.

Discretisation. A quad-tree decomposition of the feature space inherently implies that we can sample the actual averages and deviations at radii that are proportional to powers of two (or, in general, c^l multiples of r_{min}, for some integers c and l). In essence, we discretise all quantities involved by sampling them at intervals of size 2^l. However, perhaps surprisingly, this discretisation does not have a significant impact on our ability to detect outliers. Consider a relatively isolated object p_i and a distant cloud of objects. Recall that we compute MDEF values for an object starting with the smallest radius for which its sampling neighbourhood has $n_{min} = 20$ objects, in order to make the (exact) LOCI algorithm more robust and self-adapting to the local density. Similarly, for the aLOCI algorithm, we start with the smallest discretised radius for which its sampling neighbourhood has at least 20 neighbours. Considering our point p_i, observe that at large enough radius, both its sampling and counting neighbourhoods will contain many objects from the cloud, and these points will have similar neighbourhood counts to p_i, resulting in an MDEF near zero (i.e., no outlier detection). However, at some previous scale, the sampling neighbourhood will contain part of the cloud but the counting neighbourhood will not, resulting in an MDEF near one, as desired for outlier detection. Note that, in order for this to work, it is crucial that (a) we use an $\alpha \leq 2^{-l}$, and (b) we perform n_{min} neighbourhood thresholding based on the sampling neighbourhood and not the counting neighbourhood.

Locality. Ideally, we would like to have the quad-tree grids contain each object of the dataset at the exact centre of cells. This is not possible, unless we construct one quad-tree per object, which is ridiculously expensive. However, a single grid may provide a close enough approximation for many objects in the data set. Furthermore, outstanding outliers are typically detected no matter what the grid positioning is: the further an object is from its neighbours, the more "leeway" we have to be off-centre (by up to at least half the distance to its closest neighbour!).

In order to further improve accuracy for less obvious outliers, we utilise several grids. In practice, the number of grids g does not depend on the feature space dimension k, but rather on the distribution of objects (or, the *intrinsic* dimensionality [CNBYM01, BF95] of the data set, which is typically much smaller than k). Thus, in practice, we can achieve good results with a small number of grids.

To summarise, the user may select g depending on the desired accuracy vs. speed. Outstanding outliers are typically caught regardless of grid alignment. Performance on less obvious outliers can be significantly improved using a small number $g - 1$ of extra grids.

Next we have to answer two related questions: how should we pick grid alignments and,

given the alignments, how should we select the appropriate grid for each point?

Grid alignments. Each grid is constructed by shifting the quad-tree bounding box by s (a k-dimensional vector)[3]. At each grid level l (corresponding to cell diameter $d_l = R_{\mathbb{P}}/2^l$), the shift effectively "wraps around," i.e., each cell is effectively shifted by $s \mod d_l$, where $\mod$ is applied element-wise and should be interpreted loosely (as the fractional part of the division). Therefore, with a few shifts (each portion of significant digits essentially affecting different levels), we can achieve good results throughout all levels. In particular, we recommend using shifts obtained by selecting each coordinate uniformly at random from its domain.

Grid selection. For any object p_i in question, which cells and from which grids do we select to (approximately) cover the counting and sampling neighbourhoods? For the counting neighbourhood of p_i, we select a cell C_i (at the appropriate level l) that contains p_i as close as possible to its centre; this can be done in $O(kg)$ time.

For the sampling neighbourhood, a naive choice might be to search all cells in the *same* grid that are adjacent to C_i. However, the number of such cells is $O(2^k)$, which leads to prohibitively high computational cost for high dimensional data. Unfortunately, if we insist on this choice, this cost cannot be avoided; we will either have to pay it when building the quad-tree or when searching it.

Instead, we select a cell C_j of diameter d_l/α (where $d_l = R_{\mathbb{P}}/2^l$) in some grid (possibly a different one), such that the centre of C_j lies as close as possible to the centre of C_i. The reason we pick C_j based on its distance from the centre of C_i and *not* from p_i is that we want the maximum possible volume overlap of C_i and C_j. Put differently, we have already picked an approximation for the counting neighbourhood of p_i (however good or bad) and next we want the best approximation of the sampling neighbourhood, *given* the choice of C_i. If we used the distance from p_i we might end up with the latter approximation being "incompatible" with the former. Thus, this choice is the one that gives the best results. The final step is to estimate MDEF and σ_{MDEF}, by performing a box-count on the sub-cells of C_j.

Deviation estimation. A final important detail has to do with successfully estimating σ_{MDEF}. In certain situations (typically, in either very small or very large scales), many of the sub-cells of C_j may be empty. If we do a straight box-count on these, we may under-estimate the deviation and erroneously flag objects as outliers.

[3]Conceptually, this is equivalent to shifting the entire data set by $-s$

Dataset	Description
`Dens`	Two 200-point clusters of different densities and one outstanding outlier.
`Micro`	A micro-cluster with 9 points, a large, 600-point cluster (same density) and one outstanding outlier.
`Sclust`	A Gaussian cluster with 500 points.
`Multimix`	A 250-point Gaussian cluster, two uniform clusters (200 and 400 points), three outstanding outliers and 3 points along a line from the sparse uniform cluster.
`NBA`	Games, points per game, rebounds per game, assists per game (1991–92 season).
`NYWomen`	Marathon runner data, 2229 women from the NYC marathon: average pace (in minutes per mile) for each stretch (6.2, 6.9, 6.9 and 6.2 miles)

Table 3.2: Description of synthetic and real data sets.

This problem is essentially solved by giving more weight to the counting neighbourhood of p_i: in the set of box counts used for $S_q(p_i, r, \alpha)$, we also include c_i w times ($w = 2$ works well in all the datasets we have tried), besides the counts for the sub-cells of C_j.

Lemma 4 (Deviation smoothing). *If we add a new value a to set of N values with average m and variance s^2, then the following hold about the new average μ and variance σ^2:*

$$\sigma^2 > s^2 \Leftrightarrow \frac{|a-m|}{s} > \frac{N+w}{N} \quad \textit{and} \quad \lim_{N\to\infty} \frac{\sigma^2}{s^2} = 1$$

where w is the weight of a (i.e., it is counted w times).

Proof. From the definitions for mean and standard deviation, we have

$$\mu = \frac{w}{N+w}a + \frac{N}{N+w}m, \quad \sigma^2 = \frac{w}{N+w}(a-\mu)^2 + \frac{N}{N+w}s^2$$
$$\text{and} \quad (a-\mu)^2 = \left(\frac{N}{N+w}\right)^2 (a-m)^2$$

Therefore $\frac{\sigma^2}{s^2} = \frac{N^2}{(N+w)^3}\left(\frac{a-m}{s}\right)^2 + \frac{N}{N+w}$. The results follow from this relation. □

From Lemma 4, if the number of non-empty sub-cells is large, a small w weighting has small effect. For outstanding outliers (i.e., large $|a-m|/s$), this weighting does not affect the the estimate of σ_{MDEF} significantly. Thus, we may only err on the conservative side for a few outliers, while avoiding several "false alarms" due to underestimation of σ_{MDEF}.

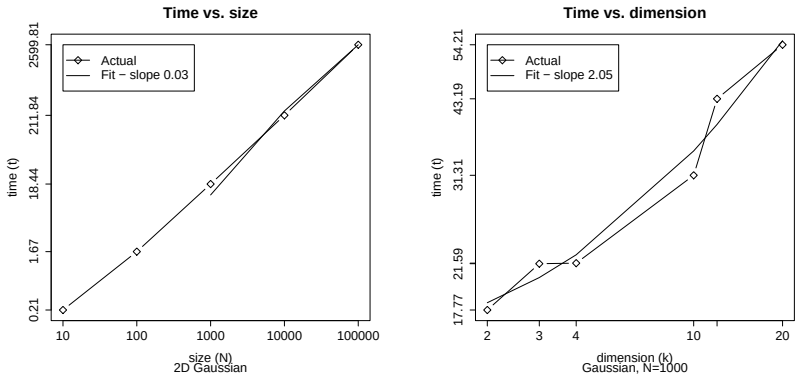


Figure 3.7: Time versus data set size and dimension (log-log scales).

3.4.2 The approximation algorithm

The aLOCI algorithm, based on the discussion in the previous section, is illustrated in Figure 3.6. The quad-tree construction stage takes time $O(NLkg)$, where L is the total number of levels (or scales), i.e., $O(\lg(r_{max}/r_{min}))$. The scoring and flagging stage takes an additional $O(NL(kg + 2^k)$ time (recall that α is a constant). As noted above, the number of grids g depends on the intrinsic dimensionality of $\mathbb{P}$. We found $10 \leq g \leq 30$ sufficient in all our experiments. Similarly, L can be viewed as fixed for most data sets. Finally, the 2^k term is a pessimistic bound because of the sparseness in the box counts. As shown in Section 3.5, in practice the algorithm scales linearly with data size and with dimensionality. Moreover, even in the worst case, it is asymptotically significantly faster than the best previous density-based approach.

3.5 Experimental evaluation

In this section we discuss results from applying our method to both synthetic and real datasets (described in Table 3.2). We also briefly discuss actual performance measurements (wall-clock times).

3.5.1 Complexity and performance

Our prototype system is implemented in Python, with Numerical Python for fast matrix manipulation and certain critical components (quad-trees and distance matrix computation) implemented in C as language extensions (achieving a 5× to 15× speedup). We are currently re-implementing the system in C and preliminary results show at least a 10× overall

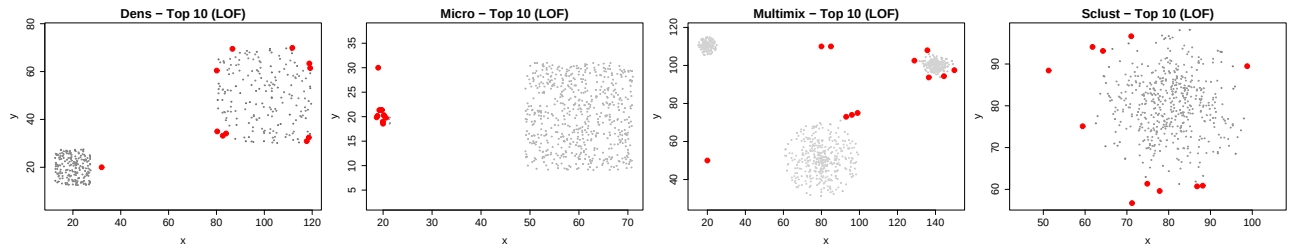


Figure 3.8: Synthetic data: LOF ($MinPts = 10$ to 30, top 10).

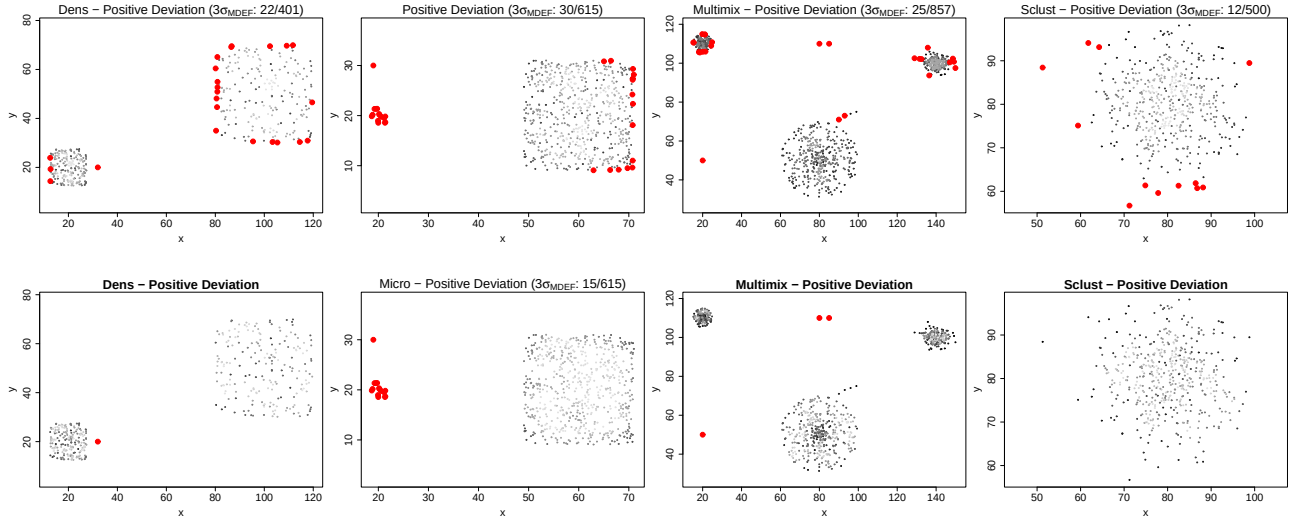


Figure 3.9: Synthetic, LOCI. Top row: $\hat{n} = 20$ to full radius, $\alpha = 0.5$. Bottom row: $\hat{n} = 20$ to 40 except `micro` where $\hat{n} = 200$ to 230, $\alpha = 0.5$.

speedup. Figure 3.7 shows the wall clock times on a synthetic dataset, versus data set size and dimension. All experiments were run on a PII 350MHz with 384Mb RAM. The graphs clearly show that aLOCI scales linearly with dataset size as well as dimension, as expected. In should be noted that the dataset chosen (a multi-dimensional Gaussian cluster) is actually much denser throughout than a real dataset would be. Thus, the time vs. dimension results are on the conservative side ($l_\alpha = 4$, or $\alpha = 1/16$ in our experiments).

3.5.2 Synthetic data

We illustrate the intuition behind LOCI using a variety of synthetic datasets, demonstrate that LOCI and aLOCI provide sound and useful results and we discuss how to interpret LOCI plots "in action." The results from LOF are shown in Figure 3.8. LOF is the current state of the art in outlier detection. However, it provides no hints about how high an outlier score is high enough. A typical use of selecting a range of interest and examining the top-N

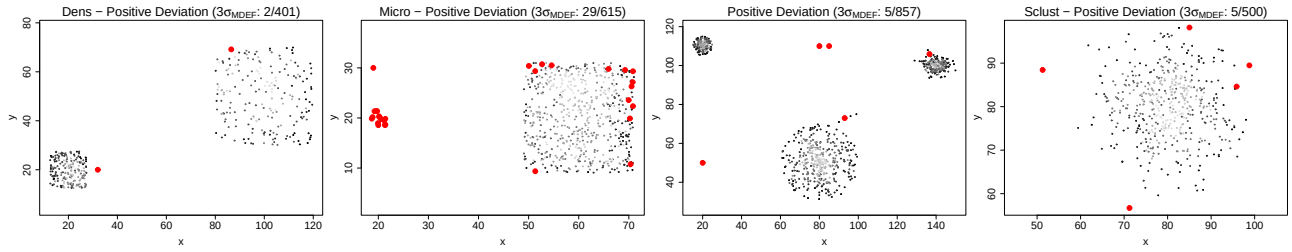


Figure 3.10: Synthetic: aLOCI (10 grids, 5 levels, $l_\alpha = 4$, except `micro`, where $l_\alpha = 3$).

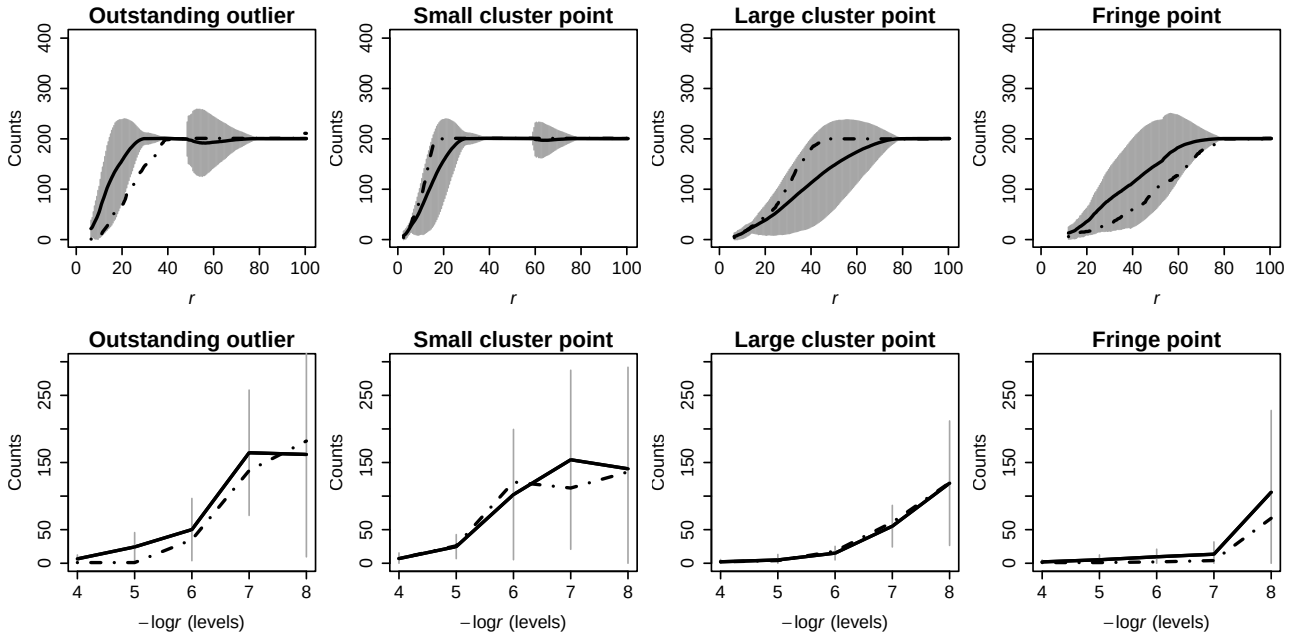


Figure 3.11: `Dens`, LOCI plots (top row) and aLOCI plots (bottom row).

scores will either erroneously flag some points (N too large) or fail to capture others (N too small). LOCI provides an automatic way of determining outliers within the range of interest and captures outliers correctly.

Figure 3.9 shows the results from LOCI on the entire range of scales, from 20 to $R_{\mathbb{P}}$ on the top row. On the bottom row, we show the outliers at a subset of that range (20 to 40 neighbours around each point). The latter is much faster to compute, even exactly, and still detects the most significant outliers. Finally, Figure 3.10 shows the aLOCI results. However, LOCI does not stop there and can provide information about *why* each point is an outlier and about its vicinity (see Figure 3.12 and Figure 3.11).

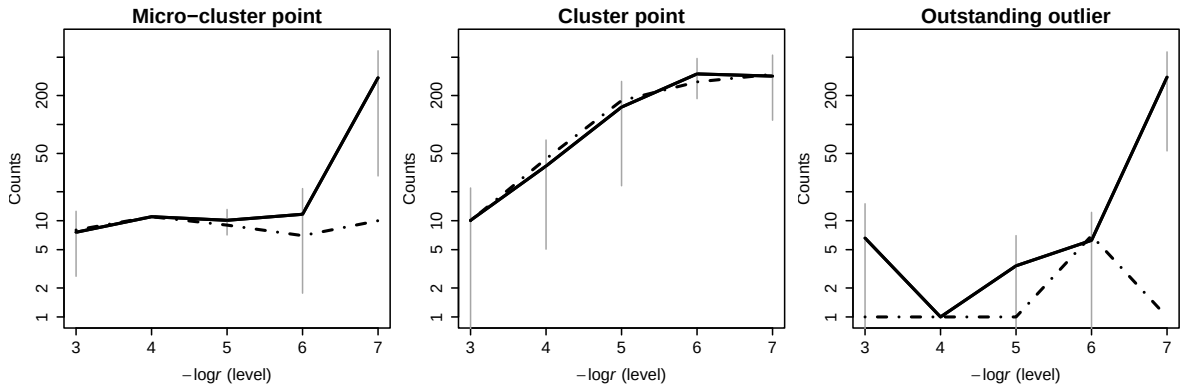


Figure 3.12: `Micro`, aLOCI plots—see Figure 3.4 for corresponding LOCI plots.

`Dens` dataset. LOCI captures the outstanding outlier. By examining the LOCI plots we can get much more information. In the leftmost column of Figure 3.11 it is clear that the outstanding outlier is indeed significantly different from its neighbours. Furthermore, the radius where the deviation first increases (≈ 5) and the associated jumps in counts correspond to the distance ($\approx 5/2$) to the first cluster. The deviation increase (without change in counts) in the range of 50–80 corresponds to the diameter (≈ 30) of the second cluster.

The second column in Figure 3.11 shows a point in the micro-cluster, which behaves very similarly to those in its sampling neighbourhood. Once again, the deviation increases correspond to the diameters of the two clusters.

Finally, the two rightmost columns of Figure 3.11 show the LOCI plots for two points in the large cluster, one of them on its fringe. From the rightmost column it is clear that the fringe point is tagged as an outlier at a large radius and by a small margin. Also, the width of the radius range with increased deviation corresponds to the radius of the large cluster.

"Drill-down." It is important to note that the aLOCI plots (bottom row) already provide much of the information contained in the LOCI plots (top row), such as the scale (or radius range) at which each point is an outlier. *If* users desire detailed information about a particular range of radii, they can select a few points flagged by aLOCI and obtain the LOCI plots. Such a "drill-down" operation is common in decision support systems. Thanks to the accuracy of aLOCI, the user can immediately focus on just a few points. Exact computation of the LOCI plots for a handful of points is fast (in the worst case—i.e., full range of radii—it is $O(kN)$ with a very small hidden constant; typical response time is about one to two minutes on real datasets).

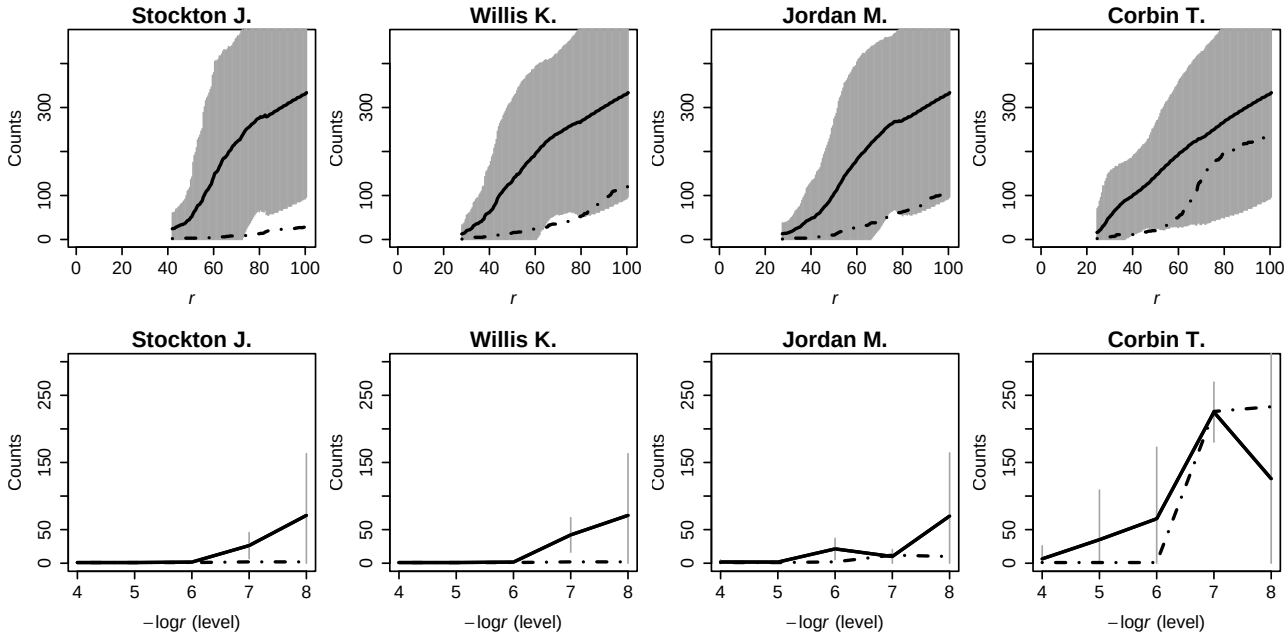


Figure 3.13: NBA, LOCI plots (top row) and aLOCI plots (bottom row).

Micro dataset. In the micro dataset, LOCI automatically captures *all* 14 points in the micro-cluster, as well as the outstanding outlier. At a wider range of radii, some points on the fringe of the large cluster are also flagged. The LOCI and aLOCI plots are in Figure 3.4 and Figure 3.12, respectively (see Section 3.2.4 for discussion).

Sclust and Multimix datasets. We discuss these briefly—the LOCI plots are similar to those already discussed, or combinations thereof. In the sclust dataset, as expected, for small radii we do not detect any outliers, whereas for large radii we capture some large deviants. Finally, in the multimix dataset, LOCI captures the isolated outliers, some of the "suspicious" ones along the line extending from the bottom uniform cluster and large deviants from the Gaussian cluster.

3.5.3 Real data

In this section we demonstrate how the above rules apply in a real dataset (see Table 3.2). In the previous section we discussed the shortcomings of other methods that provide a single number as an "outlier-ness" score. We only show LOCI and aLOCI results and discuss the LOCI plots from one real dataset (more results are in the full version of the thesis).

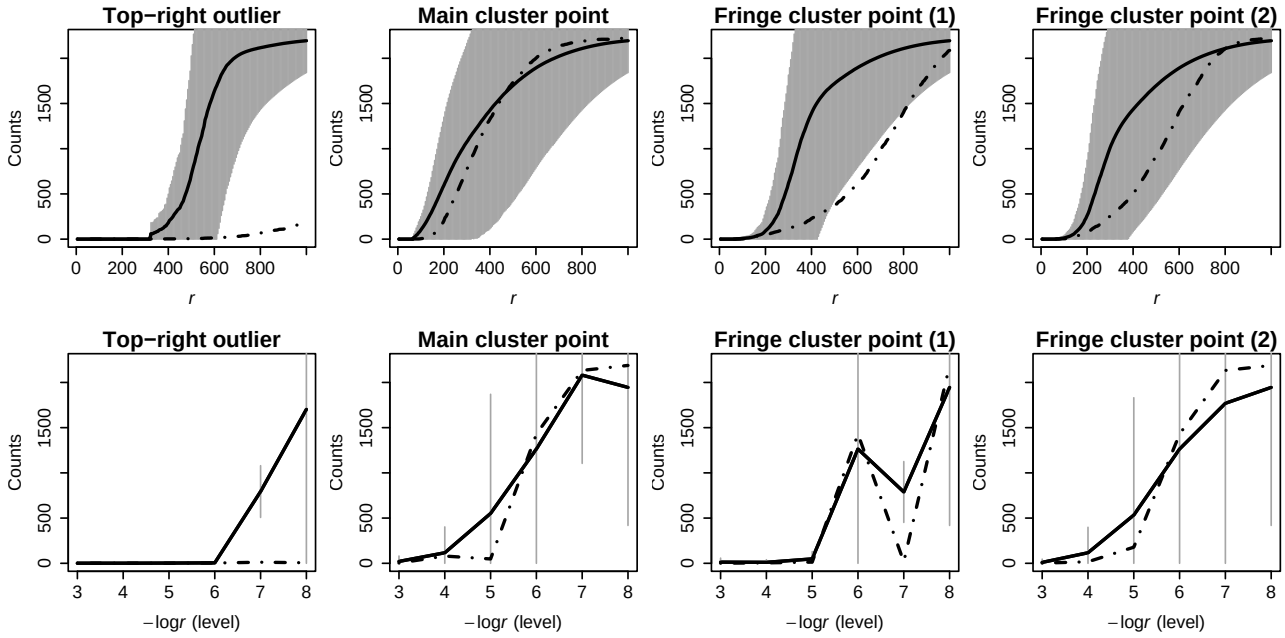


Figure 3.14: `NYWomen`, LOCI plots (top row) and aLOCI plots (bottom row).

`NBA` dataset. Results from LOCI and aLOCI are shown in Figure 3.15 (for comparison, see Table 3.3). Figure 3.13 shows the LOCI plots. The overall deviation indicates that the points form a large, "fuzzy" cluster, throughout all scales. Stockton is clearly an outlier, since he is far different from all other players, with respect to *any* statistic. Jordan is an interesting case; although he is the top-scorer, there are several other players whose overall performance is close (in fact, Jordan does not stand out with respect to any of the other statistics). Corbin is one of the players which aLOCI misses. In Figure 3.15 he does not really stand out. In fact, his situation is similar to that of the fringe points in the `Dens` dataset!

`NYWomen` dataset. Results from LOCI are shown in Figure 3.16 (aLOCI provides similar results). This dataset also forms a large cluster, but the top-right section of the cluster is much less dense than the part containing the vast majority of the runners. Although it may initially seem surprising, upon closer examination, the situation here is very similar to the `Micro` dataset! There are two outstanding outliers (extremely slow runners), a sparser but significant "micro-cluster" of slow/recreational runners, then the vast majority of "average" runners which slowly merges with an equally tight (but smaller) group of high-performers. Another important observation is that the fraction of points flagged by both LOCI and aLOCI (about 5%) is well within our expected bounds. The LOCI plots are shown in Figure 3.14 and can be interpreted much like those for the `Micro` dataset.

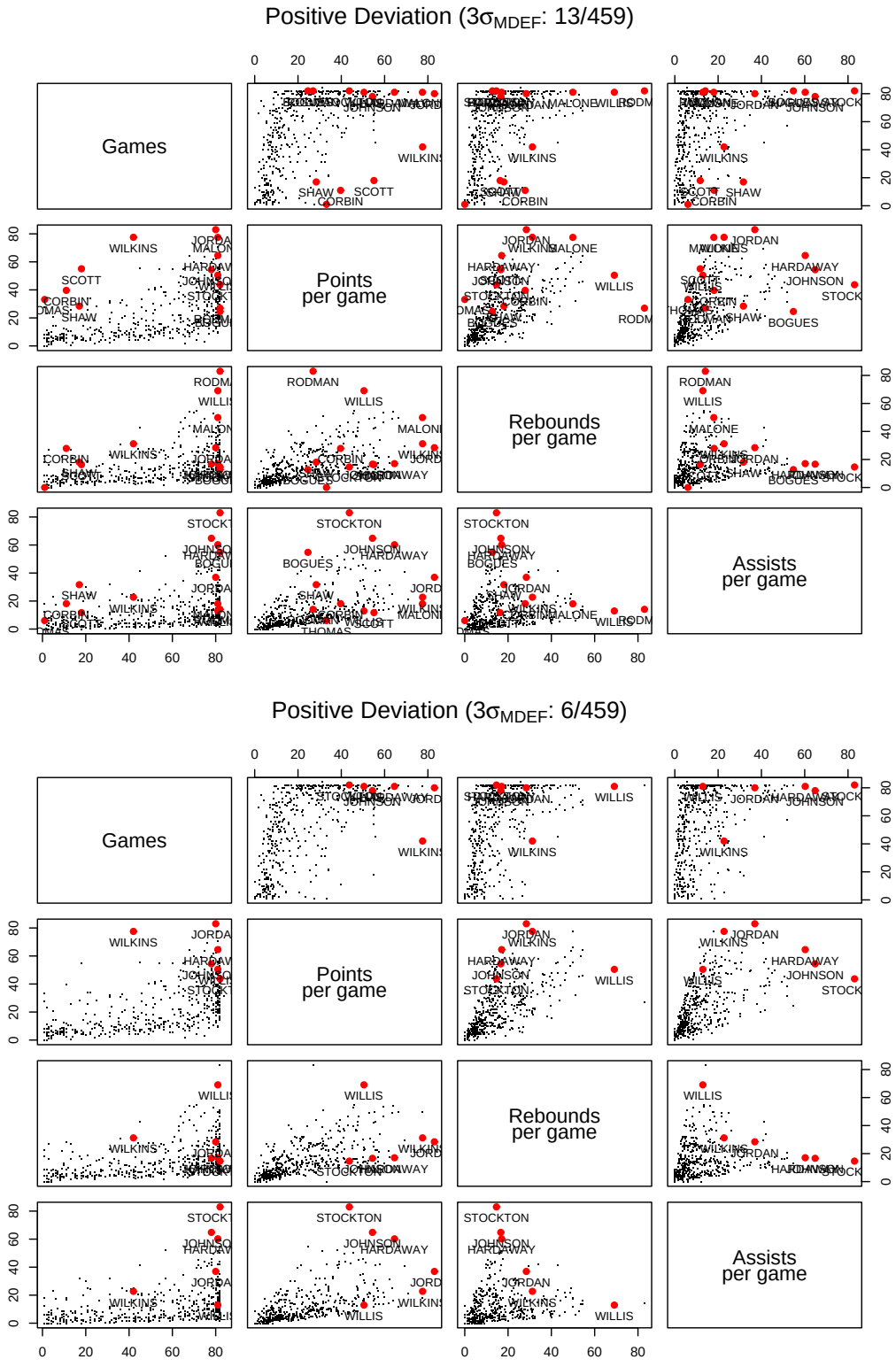


Figure 3.15: NBA results, LOCI ($\hat{n} = 20$ to full radius) and aLOCI (bottom; 5 levels, $l_\alpha = 4$, 18 grids).

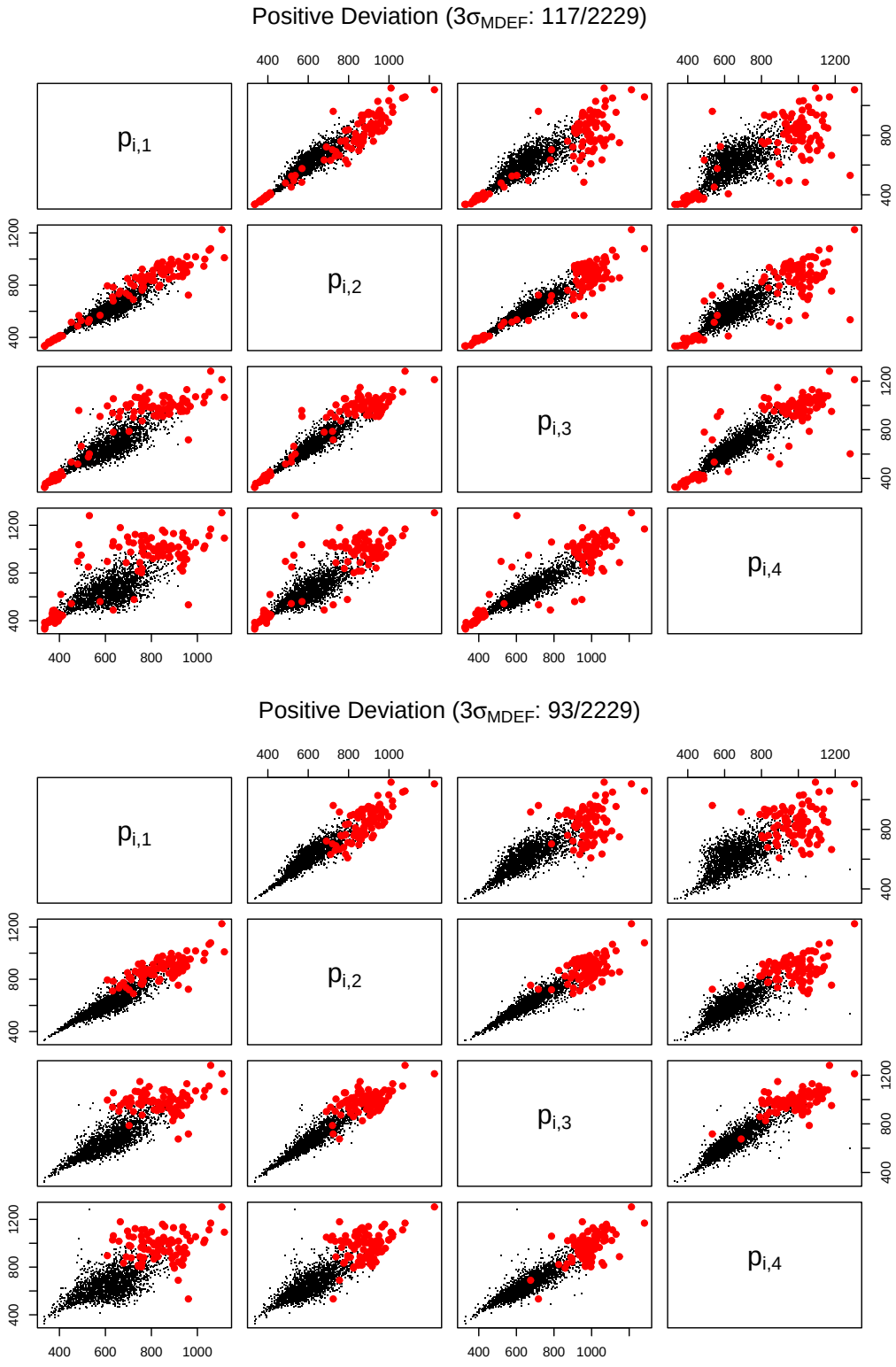


Figure 3.16: `NYWomen`, results, LOCI ($\hat{n} = 20$ to full radius) and aLOCI (bottom; 6 levels, $l_\alpha = 3$, 18 grids).

LOCI		aLOCI		LOCI		aLOCI	
#	**Player**	**#**	**Player**	**#**	**Player**	**#**	**Player**
1	Stockton J. (UTA)	1	Stockton J (UTA)	8	Corbin T. (MIN)		
2	Johnson K. (PHO)	2	Johnson K (PHO)	9	Malone K. (UTA)		
3	Hardaway T. (GSW)	3	Hardaway T (GSW)	10	Rodman D. (DET)		
4	Bogues M. (CHA)			11	Willis K. (ATL)	6	Willis K (ATL)
5	Jordan M. (CHI)	4	Jordan M (CHI)	12	Scott D. (ORL)		
6	Shaw B. (BOS)			13	Thomas C.A. (SAC)		
7	Wilkins D. (ATL)	5	Wilkins D (ATL)				

Table 3.3: NBA outliers with LOCI and aLOCI. All aLOCI outliers are shown in this table; see also Figure 3.15.

3.6 Conclusions

In summary, the main contributions of LOCI are:

- Like the state of the art, it can detect outliers and groups of outliers (or, micro-clusters). It also includes several of the previous methods (or slight variants thereof) as a "special case."
- Going beyond any previous method, it proposes an automatic, data-dictated cut-off to determine whether a point is an outlier—in contrast, previous methods let the users decide, providing them with no hints as to what cut-off is suitable for each dataset.
- Our method successfully deals with both local density and multiple granularity.
- Instead of just an "outlier-ness" score, it provides a whole plot for each point that gives a wealth of information.
- Our exact LOCI method can be computed as quickly as previous methods.
- Moreover, LOCI leads to a very fast, practically linear approximate algorithm, *aLOCI*, which gives accurate results. To the best of our knowledge, this is the first time approximation techniques have been proposed for outlier detection.
- Extensive experiments on synthetic and real data show that LOCI and aLOCI can automatically detect outliers and micro-clusters, without user-required cut-offs, and that they quickly spot outliers, expected and unexpected.

Chapter 4

Outliers by example

The notion of what is an outlier (or, exceptional/abnormal object) varies among users, problem domains and even datasets (problem instances): (i) different users may have different ideas of what constitutes an outlier, (ii) the same user may want to view a dataset from different "viewpoints" and, (iii) different datasets do not conform to specific, hard "rules" (if any). We consider objects that can be represented as multi-dimensional, numerical tuples. Such datasets are prevalent in several applications. From a general perspective [4,7,8,2], an object is, intuitively, an outlier if it is in some way "significantly different" from its "neighbours." Different answers to what constitutes a "neighbourhood," how to determine "difference" and whether it is "significant," would provide different sets of outliers. Typically, users are experts in their problem domain, not in outlier detection. However, they often have a few example outliers in hand, which may "describe" their intentions and they want to nd more objects that exhibit outlier-ness characteristics similar to those examples. Existing systems do not provide a direct way to incorporate such examples in the discovery process.

Example We give a concrete example to help clarify the problem. The example is on a 2-d vector space which is easy to visualise, but ideally our method should work on arbitrary dimensionality or, even, metric datasets[1].

Consider the dataset in Figure 4.1. In this dataset, there are a large sparse cluster, a small dense cluster and some clearly isolated objects. Only the isolated objects (circle dots) are outliers from a birds eye view. In other words, when we examine wide-scale neighbourhoods (i.e., with large radiuse.g., covering most of the dataset), only the isolated objects have very low neighbour densities, compared with objects in either the large or the small cluster. How-

[1] A metric dataset consists of objects for which we only know the pairwise distances (or, "similarity"), without any further assumptions.

ever, consider the objects on the fringe of the large cluster (diamond dots). These can also be regarded as outliers, if we look closer at mid-scale (i.e., radius) neighbourhoods. Also, objects on the fringe of the small cluster (cross dots) become outliers, if we further focus into small-scale neighbourhoods. As exemplified here, different objects may be regarded as outliers, depending on neighbourhood scale (or, size). This scenario is intuitive from the users perspective. However, to the best of our knowledge, none of the existing methods can directly incorporate user examples in the discovery process to nd out the hidden outlier concept that users may have in mind.

In this chapter, we propose *Outlier By Example (OBE)*, an outlier detection method that can do precisely that: discover the desired outlier-ness at the appropriate scales, based on a small number of examples. There are several challenges in making this approach practical; we briefly list the most important: (1) What are the appropriate features that can capture outlier-ness? These should ideally capture the important characteristics concisely and be efficient to compute. However, feature selection is only the tip of the iceberg. (2) Furthermore, we have to carefully choose exactly what features to extract. (3) The method should clearly require minimal user input and effectively use a small number of positive examples in order to be practical. Furthermore, it should ideally not need negative examples. (4) Given these requirements, can we train a classier using only the handful of positive examples and unlabelled data? In this chapter we describe the key algorithmic challenges and design decisions in detail.

In summary, the main contributions in this chapter are: (1) We introduce example-based outlier detection. (2) We demonstrate its intuitiveness and feasibility. (3) We propose OBE, which, to the best of our knowledge, is the first method to provide a solution to this problem. (4) We evaluate OBE on both real and synthetic data, with several small sets of user examples. Our experiments demonstrate that OBE can successfully incorporate these examples in the discovery process and detect outliers with outlier-ness characteristics very similar to the given examples.

The remainder of this chapter is organised as follows: In Section 4.1, we discuss related work on outlier detection. In Section 4.2, we discuss the measurement of outlier-ness and the different properties of outliers. Section 4.3 presents the proposed method in detail. Section 4.4 reports the extensive experimental evaluation on both synthetic and real datasets. Finally, in Section 4.5 we conclude.

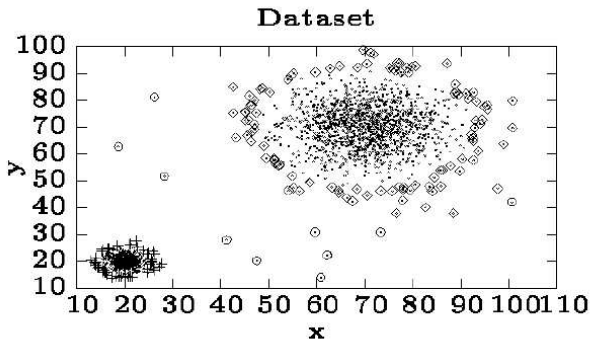


Figure 4.1: Illustration of different kinds of outliers in a dataset.

4.1 Related work

In essence, outlier detection techniques traditionally employ unsupervised learning processes. The several existing approaches are reviewed in Section 3.1. In Chapter 3 we proposed the multi-granularity deviation factor (MDEF) and LOCI. MDEF measures the "outlier-ness" of objects in neighbourhoods of different scales. LOCI examines the MDEF values of objects in all ranges and flags as outliers those objects whose MDEF values deviate significantly from the local average in neighbourhoods of some scales. The definition of MDEF can capture "outlier-ness" in different scales and the default outlier flagging criteria provide useful and meaningful results. However, users may still have to manually examine deviations at different scales. In this chapter we explore how automate things further, so the user is never exposed to any parameters.

Another outlier detection method was developed in [YT01], which focuses on the discovery of rules that characterise outliers, for the purposes of filtering new points in a security monitoring setting. This is a largely orthogonal problem. Outlier scores from SmartSifter are used to create labelled data, which are then used to find the outlier filtering rules.

In summary, all the existing methods are designed to detect outliers based on some prescribed criteria for outliers. To the best of our knowledge, this is the first proposal for outlier detection using user-provided examples.

4.2 Measuring outlier-ness

In order to understand the users' intentions and the outlier-ness they are interested in, a first, necessary step is measuring the outlier-ness. It is crucial to select features that capture the important characteristics concisely. However, feature selection is only the initial step.

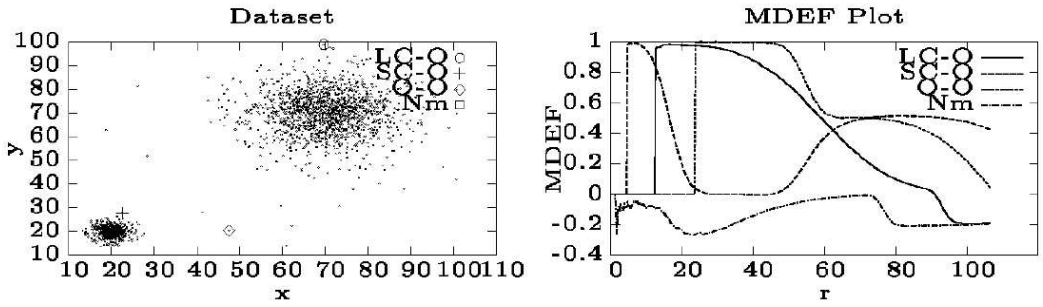


Figure 4.2: Illustrative dataset and MDEF plots.

In OBE, we employ MDEF for this purpose, which measures outlier-ness of objects in the neighbourhoods of different scales (i.e., radii).

Detailed definition of the multi-granularity deviation factor (MDEF) is given in Section 3.2. Here we repeat some basic terms and notation for completeness. Let the r-neighbourhood of an object p_i be the set of objects within distance r of p_i . Let $n(p_i, \alpha r)$ and $n(p_i, r)$ be the numbers of objects in the αr-neighbourhood (counting neighbourhood) and r-neighbourhood (sampling neighbourhood) of p_i, respectively[2]. Let $\hat{n}(p_i, r, \alpha)$ be the average, over all objects p in the r-neighbourhood of p_i of $n(p, \alpha r)$.

Definition 11 (MDEF). *For any p_i , r and α, the multi-granularity deviation factor (MDEF) at radius (or scale) r is defined as follows:*

$$\text{MDEF}(p_i, r, \alpha) = \frac{\hat{n}(p_i, r, \alpha) - n(p_i, \alpha r)}{\hat{n}(p_i, \alpha, r)}$$

Intuitively, the MDEF at radius r for a point p_i is the relative deviation of its local neighbourhood density from the average local neighbourhood density in its r-neighbourhood. Thus, an object whose neighbourhood density matches the average local neighbourhood density will have an MDEF of 0. In contrast, outliers will have MDEFs far from 0. The MDEF values are examined (or, sampled) at a wide range of sampling radii r, $r_{min} \leq r \leq r_{max}$, where r_{max} is the maximum distance of all object pairs in the given dataset and r_{min} is determined based on the number of objects in the r-neighbourhood of p_i . In our experiments, for each p_i in the dataset, r_{min} for p_i (denoted by $r_{min,i}$) is the distance to its 20-th nearest neighbour. In other words, we do not examine the MDEF value of an object until the number of objects in its sampling neighbourhood reaches 20. This is a reasonable choice which effectively avoids

[2]In all experiments, $\alpha = 0.5$, as in Chapter 3 and [PKGF03].

introduction of statistical errors in MDEF estimates.

Next we give some examples to better illustrate MDEF. Figure 4.2 shows a dataset which has mainly two groups: a large, sparse cluster and a small, dense one, both following a Gaussian distribution. There are also a few isolated points. We show MDEF plots for four objects in the dataset.

- Consider the point Nm in the middle of the large cluster (at about $x = 70, y = 68$). The MDEF value is low at all scales: compared with its neighbourhood, whatever the scale is, the local neighbourhood density is always similar to the average local density in its sampling neighbourhood. So, the object can be always regarded as a normal object in the dataset.

- In contrast, for the other three objects, there exist situations where the MDEFs are very large, some times even approaching 1. This shows that they differ significantly from their neighbours in some scales. The greater the MDEF value is, the stronger the degree of outlier-ness.

Even though all three objects in Figure 4.2 can be regarded as outliers, they are still different, in that they exhibit outlier-ness at different scales.

- The MDEF value of the outlier in the small cluster, SC-O, (at about $x = 22, y = 27$), reaches its maximum at radius $r \approx 5$, then it starts to decrease rapidly until it becomes 0 and remains there for a while (in the range of $r \approx 23 - -45$). Then the MDEF value increases again but only to the degree of 0.6. The change of MDEF values indicates that the object is extremely abnormal compared with objects in the very small local neighbourhood (objects in the small cluster).

- On the other hand, the outlier of the large cluster, LC-O, (at about $x = 70, y = 98$), exhibits strong outlier-ness in the range from $r = 10$ to $r = 30$, then becomes more and more ordinary as we look at a larger scale.

- For the isolated outlier, O-O, (at about $x = 47, y = 20$), its MDEF value stays at 0 up to almost $r = 22$, indicating that it is an isolated object. Then, it immediately displays a high degree of outlier-ness.

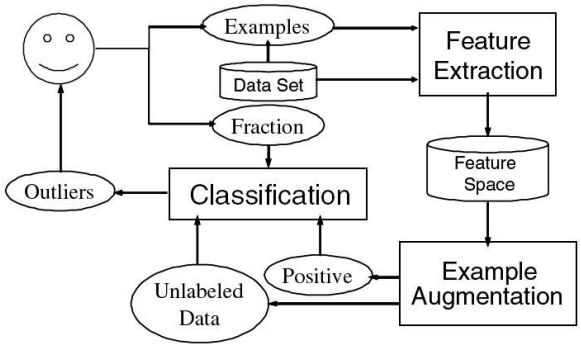


Figure 4.3: The framework of OBE.

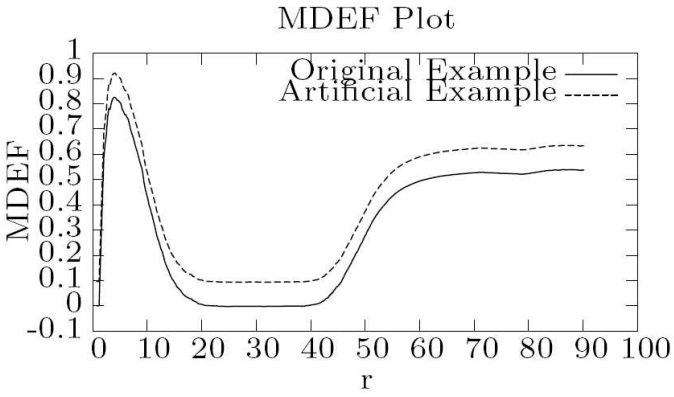


Figure 4.4: The artificial and the original examples.

4.3 Proposed method (OBE)

OBE detects outliers based on user-provided examples and a user-specified fraction of objects to be detected as outliers in the dataset. OBE performs outlier detection in three stages: feature extraction step, example augmentation step and classification step. Figure 4.3 shows the overall OBE framework.

4.3.1 Feature extraction step

The purpose of this step is to map all objects into the MDEF-based feature space, where the MDEF plots of objects capturing the degree of outlier-ness, as well as the scales at which the outlier-ness appears, are represented by vectors. Let D be the set of objects in the feature space. In this space, each object is represented by a vector: $O_i := (m_{i0}, m_{i1}, \ldots, m_{in},$ where

$m_{ij} := \text{MDEF}(p_i, r_j, \alpha r)$, $0 \leq j \leq n$ and $r_0 := \min_k(r_{min,k})$, $r_n := r_{max}$, $r_j = \frac{r_n - r_0}{n} j + r_0$.

4.3.2 Example augmentation step

In the context of outlier detection, outliers are usually few, and the number of examples that users could offer is even less. If we only learn from the given examples, the information is very little to be used to construct an accurate classifier. However, example-based outlier detection is practical only if the number of required examples is small. OBE effectively solves this problem by augmenting the user-provided examples.

In particular, the examples are augmented by adding outstanding outliers and artificial positive examples, based on the original examples.

Outstanding outliers After all objects are projected into the feature space, we can detect outstanding outliers. The set of outstanding outliers is defined by $\{O_i \mid max_M(O_i) > K\}$ where $max_M(O_i) := \max_j(m_{ij})$ and K is a threshold.

Artificial examples The examples are further augmented by creating "artificial" data. This is inspired by the fact that an object is sure to be an outlier if all of its feature values (i.e., MDEF values) are greater than those of the given outlier examples. Figure 4.4 shows the created artificial data and the original example.

Artificial data are generated in the following way:

1. Take the difference between $max_M(O_i)$ and the threshold K, $Diff_M(i) := K - max_M(O_i)$.

2. Divide the difference, $\text{Diff}_\text{M}(i)$ into x intervals, where x is the number of artificial examples generated from an original outlier example plus 1. For instance, if the intended augmentation ratio is 200%, two artificial example are generated from each original example. Then we divide Diff_M into 3 intervals (i.e., $x = 3$), $\text{Intv}_\text{M}(i) = \text{Diff}_\text{M}(i)/x$.

3. Then, create artificial examples as: $O_A(i,j) := (m_{i0} + j \cdot \text{Intv}_\text{M}(i), m_{i1} + j \cdot \text{Intv}_\text{M}(i), \ldots, m_{in} + j \cdot \text{Intv}_\text{M}(i))$, for $1 \leq j \leq x - 1$. Here, $O_A(i,j)$ is the j-th artificial example generated from object O_i.

In this way, the "outlier-ness strength" of the user's examples is amplified, in a way consistent with these examples.

Putting together the original examples, outstanding outliers and artificial examples, we get the positive training data.

Input:
Set of outlier examples: E
Fraction of outliers: F
Dataset: D
Output:
Outliers similar to examples
Algorithm:
$OO \leftarrow \emptyset$ // Outstanding outliers
// Feature extraction step
Foreach $p_i \in D$:
Foreach j, $1 \leq j \leq n$:
Compute MDEF value m_{ij}
If $m_{ij} > K$:
$OO \leftarrow OO \cup \{p_i\}$
// Example augmentation step
Foreach example in E:
Create artificial examples
$POS \leftarrow E \cup OO \cup$ artificial examples
// Classification step
$NEG \leftarrow$ strongest negatives
$P \leftarrow D$
Do:
$P' \leftarrow P$
SVM $\leftarrow$ TrainSVM(POS, NEG)
(P, N) $\leftarrow$ SVM.Classify(D)
$NEG \leftarrow N$
While ($|P| \geq F \cdot |D|$ and $|P| \neq |P'|$)
Return P'

Figure 4.5: The OBE procedure.

4.3.3 Classification step

So far, the (augmented) positive examples, as well as the entire, unlabelled dataset are available to us. The next crucial step is finding an efficient and effective algorithm to discover the "hidden" outlier concept that the user has in mind.

We use an SVM (Support Vector Machine) classifier to learn the outlier-ness of interest to the user and then detect outliers which match this. Traditional classifier construction needs both positive and negative training data. However, it is too difficult and also a burden for users to provide negative data. Most objects fall in this category and it is unreasonable to expect users to examine them.

However, OBE addresses this problem and can learn only from the positive examples obtained in the augmentation step and the unlabelled data (i.e., the rest of the objects in the dataset). The algorithm shown here uses the marginal property of SVMs. In this sense, it bears some general resemblance to PEBL [YHC02], which was also proposed for learning from positive and unlabelled data. However, in PEBL, the hyperplane for separating positive and negative data is set as close as possible to the set of given positive examples. In the context of OBE, the positive examples are just examples of outliers, and it is not desirable to set the hyperplane as in PEBL. The algorithm here decides the final separating hyperplane based on the fraction of outliers to be detected. Another difference between OBE and PEBL is that strong negative data are determined taking the characteristics of MDEF into

consideration.

The classification step consists of the following five sub-steps. Figure 4.5 summarises the overall procedure of OBE.

Negative training data extraction sub-step All objects are sorted in descending order of $\max_M$. Then, from the objects at the bottom of the list, we select a number of (strong) negative training data equal to the number of positive training data. Let the set of strong negative training data be *NEG*. Also, let the set of positive training data obtained in the example augmentation step be *POS*.

Training sub-step Train a SVM classifier using *POS* and *NEG*.

Testing sub-step Use the SVM to divide the dataset into the positive set P and negative set N.

Update sub-step Replace *NEG* with N, the negative data obtained in the testing sub-step.

Iteration sub-step Iterate from the training sub-step to the updating sub-step until the ratio of the objects in P converges to the fraction specified by the user. The objects in the final P are reported to the user as detected outliers.

4.4 Experimental evaluation

In this section, we describe our experimental methodology and the results obtained by applying OBE to both synthetic and real data, which further illustrate the intuition and also demonstrate the effectiveness of our method.

We use three synthetic and one real datasets (see Table 4.1 for descriptions) to evaluate OBE.

4.4.1 Experimental procedure

Our experimental procedure is as follows:

1. To simulate interesting outliers, we start by selecting objects which represent outlierness at some scales under some conditions, for instance, $\wedge_q(min_q, max_q, cond_q, K_q)$, where

Dataset	Description
`Uniform`	A 6000-point group following a uniform distribution.
`Ellipse`	A 6000-point ellipse following a Gaussian distribution.
`Mixture`	A 5000-point sparse Gaussian cluster, a 2000-point dense Gaussian cluster and 10 randomly scattered outliers.
`NYWomen`	Marathon runner data, 2229 women from the NYC marathon: average pace (in minutes per mile) for each stretch (6.2, 6.9, 6.9 and 6.2 miles).

Table 4.1: Description of synthetic and real datasets.

$(min_q, max_q, cond_q, K_q)$ stands for the condition that $(m_{ij} cond_q K_q)$ for some j such that $min_q \leq j \leq max_q$, where $cond_q$ could be either "$>$" or "$<$".

2. Then, we "hide" most of these outliers. In particular, we randomly sample y% of the outliers to serve as examples that would be picked by a user.

3. Next, we detect outliers using OBE.

4. Finally, we compare the detected outliers to the (known) simulated set of outliers. More specifically, we evaluate the success of OBE in recovering the hidden outlier concept using precision/recall measurements.

OBE reports as interesting outliers the outstanding ones, as well as those returned by the classifier. Table 4.2 shows all the sets of interesting outliers along with the corresponding discriminants used as the underlying outlier concept in our experiments. In the table, for instance, the discriminant $(1, 35, >, 0.9)$ means that objects are selected as interesting outliers when their MDEF values are greater than 0.9 in the range of radii from 1 to 35. The number of the outstanding outliers and interesting outliers is also shown in Table 4.2. We always randomly sample 10% (i.e., $y = 10$) of the interesting outliers to serve as user provided examples and "hide" the rest.

To detect outstanding outliers, we use $K = 0.97$ for all the synthetic datasets and $K = 0.99$ for the NYWomen dataset. The discovered outstanding outliers of the synthetic datasets are shown in Figure 4.6. Also, during the augmentation step, we always generate 5 (i.e., $x = 6$) artificial examples from each original example.

We use the LIBSVM [LIB] implementation for our SVM classifier. We extensively compared the accuracy of both linear and polynomial SVM kernels and found that polynomial perform consistently better. Therefore, in all experiments, we use polynomial kernels and

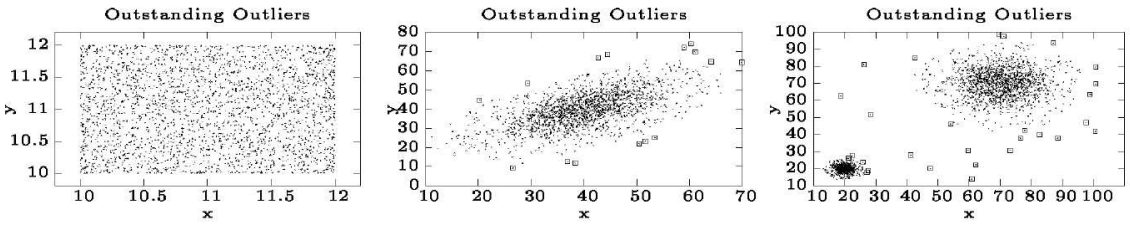


Figure 4.6: Outstanding outliers in the synthetic datasets.

the same SVM parameters[3]. Therefore, the whole processes can be done automatically. We report the effectiveness of OBE in discovering the "hidden" outliers using precision and recall measurements:

$$\text{precision} := \frac{\text{\# of correct positive predictions}}{\text{\# of positive predictions}}$$

$$\text{recall} := \frac{\text{\# of correct positive predictions}}{\text{\# of positive data}}$$

4.4.2 Results

`Uniform` dataset Figure 4.7 shows the outliers detected by OBE. Although one might argue that no objects from an (infinite!) uniform distribution should be labelled as outliers, the objects at the fringe or corner of the group are clearly "exceptional" in some sense. On the top row, we show the interesting outliers, original examples and the detected results for case U-Fringe. The bottom row shows those for case U-Corner (see Table 4.2 for a description of the cases). Note that the chosen features can capture the notion of both "edge" and "corner" and, furthermore, OBE can almost perfectly reconstruct these hidden outlier notions!

`Ellipse` dataset We simulate three kinds of interesting outliers for the ellipse dataset: (i) the set of fringe outliers whose MDEF values are examined at a wide range of scales, (ii) those mainly spread at the long ends of the ellipse which display outlier-ness in two ranges of scales (from 15 to 25 and from30 to 40), and (iii) mainly in the short ends, which do not show strong outlier-ness in the scales from 35 to 40. The output of OBE is shown in Figure 4.8. Again, the features can capture several different and interesting types of outlying

[3]For the parameter C (the penalty imposed on training data that fall on the wrong side of the decision boundary), we use 1000, i.e., a high penalty for mis-classification. For the polynomial kernel, we employ a kernel function of the form $(u' * b + 1)^2$.

objects and OBE again discovers the underlying outlier notion!

`Mixture` dataset We also mimic three categories of interesting outliers: (i) the set of outliers scattered along the fringe of both clusters, (ii) those mainly spread along the fringe of the large cluster, and (iii) those mainly in the small cluster.

`NYWomen` dataset In the real dataset, we mimic three kinds of intentions for outliers: The first group (case N-FS) is the set of consistently fast or slow runners (i.e., the fastest 7 and almost all of the 70 very slow ones). The second group of outlying runners (case N-PF) are those who are at least partly fast. In this group, we discover both the fastest 23 runners and those runners who were abnormally fast in one or two parts of the four stretches, although they rank middle or last in the whole race. For example, one of them took 47 minutes for the first 6.2 miles, while 91 minutes for the last 6.2 miles. The third set of interesting outliers (case N-SS) is those who run with almost constant speed and rank middle in the whole race. They are very difficult to perceive, but they certainly exhibit outlier-ness when we examine them at a small scale. Because of space limits, we only show the result plots in the first and fourth dimensions—see Figure 4.9.

For all datasets, Table 4.2 shows the precision and recall measurements for OBE, using polynomial kernels (as mentioned, polynomial kernels always performed better than linear kernels in our experiments). It also shows the number of iterations needed to converge in the learning step. In Table 4.2, all the measurements are averages of ten trials. In almost all cases, OBE detects interesting outliers with both precision and recall reaching 80–90%. In the worst case (case N-SS of `NYWomen`), it still achieves 66% precision and 70% recall. The number of iterations is always small (less than 10).

4.5 Conclusion

Detecting outliers is an important, but tricky problem, since the exact notion of an outlier often depends on the user and/or the dataset. We propose to solve this problem with a completely novel approach, namely, by bringing the user in the loop, and allowing him or her to give us some example records that he or she considers as outliers.

The contributions in this chapter are the following:

- We propose OBE, which, to the best of our knowledge, is the first method to provide a solution to this problem.

Dataset	*OO*	Cases				OBE		
		Label	Description	Condition	*IO*	Prec.	Recall	Iter.
`Uniform`	0	U-Fringe	Fringe	$(0.3, 0.6, >, 0.4)$	330	82.76	88.18	8.1
		U-Corner	Corner	$(1, 2, >, 0.5)$	274	91.90	97.92	4.1
`Ellipse`	15	E-Fringe	Fringe	$(5, 30, >, 0.85)$	214	90.20	93.55	6.1
		E-Long	Long ends	$(15, 25, >, 0.8)$ $(30, 40, >, 0.6)$	140	88.67	92.14	5.4
		E-Short	Short ends	$(5, 15, >, 0.8)$ $(35, 40, <, 0.6)$	169	76.46	80.00	10.4
`Mixture`	29	M-All	All	$(1, 35, >, 0.9)$	166	86.32	93.80	4.5
		M-Large	Large cluster	$(15, 35, >, 0.9)$	123	91.52	95.37	4.6
		M-Small	Small cluster	$(1, 5, >, 0.9)$	72	91.30	97.92	5.3
`NYWomen`	17	N-FS	Very fast/slow	$(900, 1400, >, 0.7)$	91	81.53	84.95	6.5
		N-PF	Partly fast	$(300, 500, >, 0.8)$ $(1400, 1600, <, 0.4)$	126	73.07	78.81	6.9
		N-SS	Stable speed	$(100, 300, >, 0.8)$ $(400, 600, <, 0.3)$ xs	121	66.55	70.74	9.2

Table 4.2: Interesting outliers, discriminants and the performance of OBE. *OO* denotes outstanding outliers, *IO* denotes interesting outliers. Precision, recall and the number of iterations for convergence in the classification step are used to show the performance of OBE.

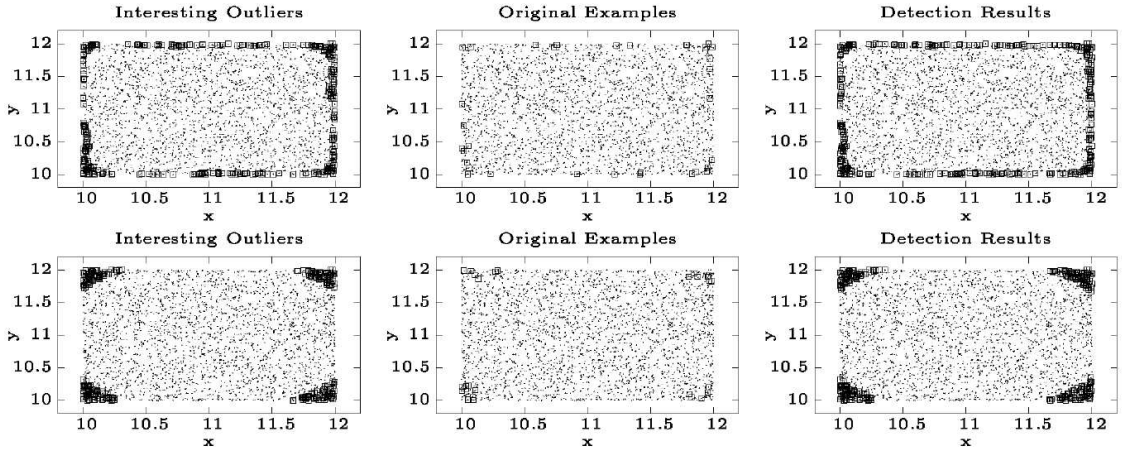


Figure 4.7: Detection results on the `Uniform` dataset. Top row: case U-Fringe, bottom row: case U-Corner—see Table 4.2 for a description of each case.

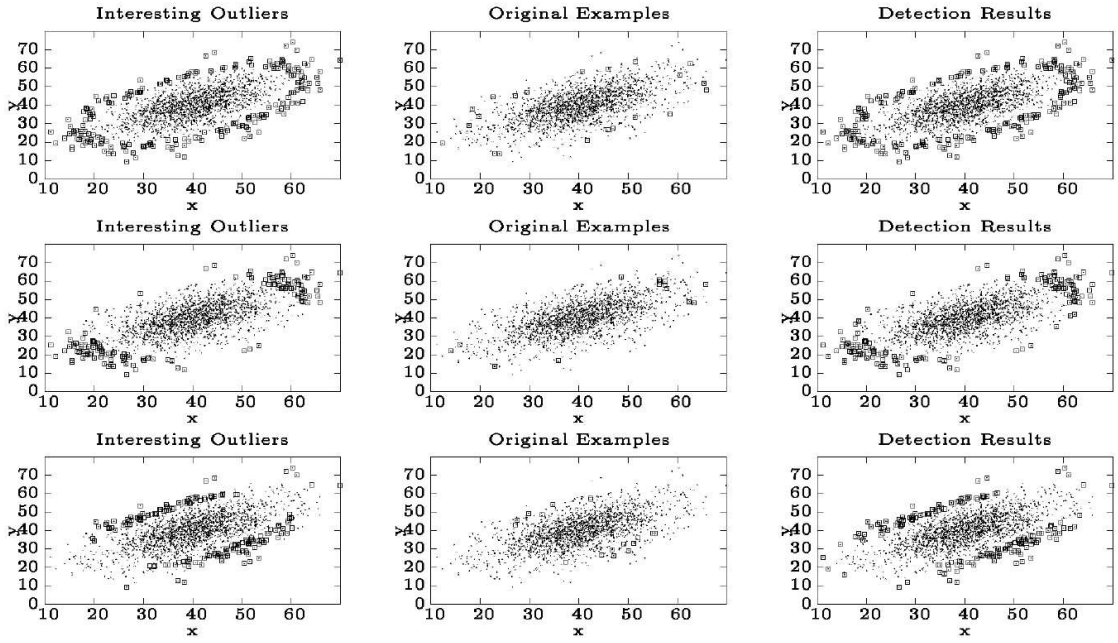


Figure 4.8: Detection results on the `Ellipse` dataset. From top to bottom row, in turn: case E-Fringe, case E-Long and case E-Short—see Table 4.2 for description of each case.

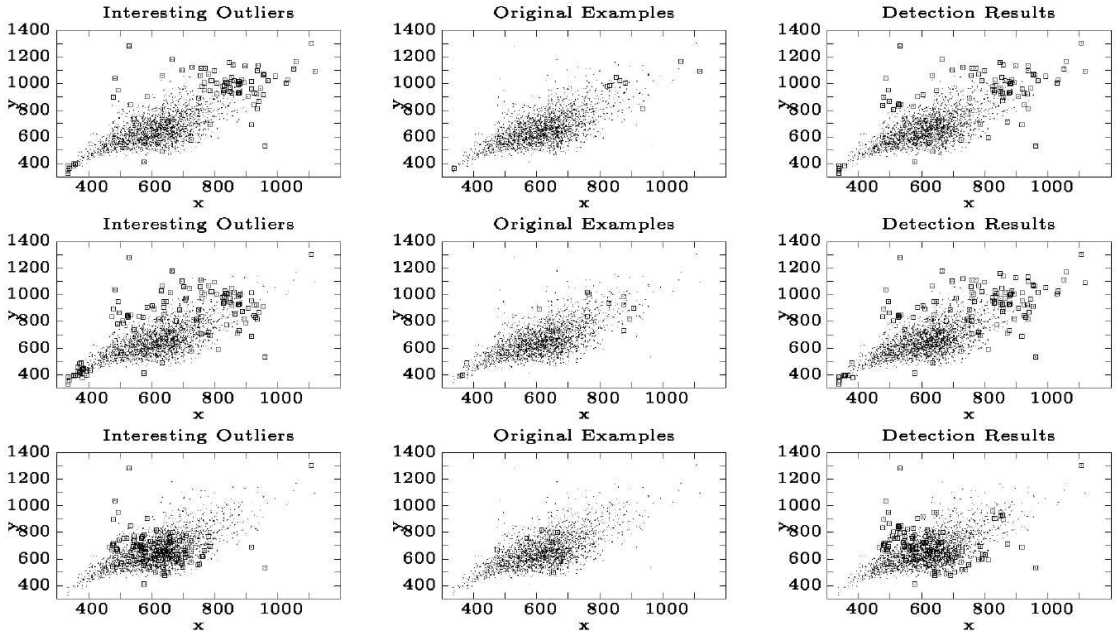


Figure 4.9: Detection results on the `NYWomen` dataset. From top to bottom row, in turn: case N-FS, case N-PF, case N-SS—see Table 4.2 for a description of each case. Only the first and fourth dimensions are used for the plots, although `NYWomen` is four-dimensional.

- We build a system, and described our design decisions. Although OBE appears simple to the user ("click on a few outlier-looking records"), there are many technical challenges under the hood. We showed how to approach them, and specifically, how to extract suitable feature vectors out of our data objects, and how to quickly train a classifier to learn from the (few) examples that the user provides.
- We evaluated OBE on both real and synthetic data, with several small sets of user examples. Our experiments demonstrate that OBE can successfully incorporate these examples in the discovery process and detect outliers with "outlier-ness" characteristics very similar to the given examples.

Chapter 5

Dimension induced clustering

Real datasets exhibit patterns and regularities. As a main consequence, points typically lie on low-dimensional manifolds, rather than being evenly spread out. Detecting subsets of points with low intrinsic dimensionality is useful in tasks such as indexing and classification. It was originally observed [PKF00] and has recently been proved [KL04] that the well-known "curse of dimensionality" translates essentially to a "curse of intrinsic dimensionality," in terms of finding efficient approximations to nearest-neighbour queries. Furthermore, separating points based on some notion of "local dimensionality" is helpful in identifying subsets of points that are qualitatively different. For example assuming a geographical setting, locations along a river belong to a 1-D manifold, whereas locations on a lake would belong to a 2-D manifold (for example Figure 5.1). Similarly, road intersections along a highway are on a 1-D manifold, while intersections within a city belong to a 2-D manifold. Thus, discovering low-dimensional manifolds is also useful in its own right.

However, we are faced with three main challenges. The first question that naturally arises is what "dimension" exactly means. Data patterns may be fairly complicated. Assuming that points follow linear trends and always lie on hyper-planes is fairly restrictive; in fact, they typically follow complex shapes, with limited extents. For example, in a more abstract setting the lake and river may lie on the same 2-D plane, but they still differ in dimension. Second, to complicate matters even further, the observations may not even belong to a vector space. Yet, we should be still able to define the dimension of a subspace. Finally, in practical applications, the dimension of the embedding space is large, in the order of thousands. Any method of practical interest should be able to deal with spaces of arbitrarily high dimension and still successfully find low-dimensional subsets embedded in the original space. It is thus desirable to characterise manifolds of complex shape embedded in any space of high dimension, and devise algorithms for identifying them. As we shall see, it is possible to

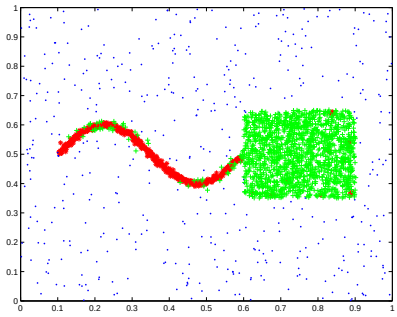


Figure 5.1: A dataset that contains two subsets of different intrinsic dimensionality

intuitively define a topological notion of dimension, which does not depend on the notion of a linear subspace. Furthermore, our algorithms for identifying low-dimensional manifolds are not sensitive to the dimension of the original space and, thus, do not suffer from the "curse of dimensionality".

In order to argue about and detect the existence of a low-dimensional manifold in our data, there must exist a sufficiently large number of points that are densely packed on this manifold. Therefore, it seems reasonable, that using density based methods it would be possible to detect such subspaces. However, we argue that density alone is not enough for this task. For the sake of examples consider city locations interspersed among highway intersections. The cities, lying on a 2-D surface, form the first cluster. The road intersections form the second cluster, as a complex network of denser 1-D lines which occupies the *same* space as the city manifold cluster. Density-based clustering approaches have a limited ability to detect clusters-within-clusters. In this simple example, they would typically produce either a large number of separate city clusters (one for each group of cities enclosed by roads), or one single cluster containing both intersections and cities, depending on the density thresholds.

Consider also the example in Figure 5.1. In this case we have three qualitatively different types of points. The set of points that lie on the curved 1-D line, the set of points that lie on the 2-D cloud, and the noise points that are scattered in the 2-D plane. If the line and the square have the same density, using a density based method is not possible to detect all three of these subsets. Any density threshold will just separate the noise points from the rest. Note also that dimensionality by itself would not be able to separate the square from the noise points, since the noise points are also 2-dimensional. The synergy of density and local dimension gives a clear separation of the three distinct datasets, as it is shown in the figure.

In this chapter we propose the idea of creating a local-growth model for each point. This growth model depends, in principle, only on pairwise point distances and captures how

each point "views" its local neighbourhood. Using this model we can characterise each point x_i with two variables (d_i, c_i), where d_i is the *local dimensionality* of the point x_i, and c_i is the *local density*. Intuitively, d_i depends on the growth rate of the number of points in the neighbourhood of x_i, while c_i depends on the density of points in the neighbourhood of x_i.

Both variables are estimated from *local growth curves*. Local growth curves can be computed directly from the data. Our algorithms require only a limited number of nearest-neighbour (NN) queries. These are well-studied and several efficient algorithms exist to answer them. For each point x_i, the local growthcurve of x_i is computed, and a line is fitted on a subset of the points of the curve that corresponds to a local neighbourhood of x_i. Then, the local dimensionality d_i is defined as the slope of the fitted line, while the local density c_i is defined as the value of the fitted line for a specific radius r^*. We choose r^* so as to maximise the information captured by the set of feature pairs (d_i, c_i), in the sense of minimising the correlation between d_i and c_i.

Using the local density and local dimensionality, each point x_i is represented by the feature pair (d_i, c_i). Therefore, we map our dataset in a two dimensional space. This has the following advantages. First, we can easily cluster the dataset using an off-the-shelf, two-dimensional clustering algorithm, like EM. Second, in an user interactive system, the number of clusters and the correct partition can be identified through visual inspection.

Our main contributions are the following:

- Drawing upon ideas from fractals, we propose a general way to characterise the *local* dimensionality of points. Our definitions are topological and independent of the notion of a linear subspace. Our methods can be applied to datasets of arbitrary dimensionality and our algorithms are independent of the number of dimensions in the original dataset.
- Our method maps the dataset into a 2-dimensional space. We show how to chose the feature pairs (d_i, c_i), so as to maximise the information they retain about the dataset, and enhance the visual representation of the dataset.
- Our algorithms can successfully detect low-dimensional manifolds embedded in high-dimensional spaces, even when they are spatially overlapping, by using the local dimensionality in addition to the local density.
- We show how we can use the local dimensionality along with local density to detect low dimensional m-flats and low-rank submatrices.

Additionally, our method does not assume that the points lie in a vector space and can

also be applied to metric data, when dimensionality is not directly obtainable from the data representation itself.

The rest of this chapter is organised as follows. Section 5.1 discusses briefly the related work. Section 5.2 introduces the key concepts and definitions. Section 5.3 explains their properties and elaborates on effectively selecting feature pairs (d_i, c_i). Section 5.4 presents our algorithms and section 5.5 applies them to the problems of detecting low-dimensional m-flats and low-rank submatrices. Finally, we conclude in section 5.6.

5.1 Related Work

In this section we briefly discuss related work, broadly divided in two categories: methods that use some notion of density, and methods based on intrinsic dimensionality.

Density-based clustering Similar to our method, density-based clustering approaches also rely on local density information in order to partition the dataset.

Hierarchical single linkage is a well-known method to find clusters with respect to density. To overcome problems in cases, when clusters are connected by small chains, popular variants like DBSCAN [EKSX96] and OPTICS [ABKS99] use a modified linkage hierarchy, where points within a cluster have to be reachable via core points (points having a certain minimum number of neighbours). DBSCAN computes a clustering corresponding to a cut in the linkage hierarchy, while OPTICS finds an ordering of the points from which a lower part of the linkage hierarchy can be deduced. Both algorithms fall short in case of clusters within clusters and the true hierarchy contains nodes with degree one. However, as our approach does not rely on spatial separation of the clusters, but focuses on detecting subsets with low intrinsic dimensionality, it can also deal with those cases.

Another density-based clustering method is DenClue [HK98], which employs kernel density estimation and uses density thresholds to define the clusters to be found. Wavecluster [SCZ98] uses the wavelet transform and can detect clusters of arbitrary shape and with different granularity. However, computing the wavelet transform in high dimensions (more than 2) is computationally challenging. CLIQUE [AGGR98] is a density-based method that can also detect subspaces such that high-density clusters exist in them. However, it is grid-based and thus assumes that points lie in vector space. Furthermore, CLIQUE considers only hyper-rectangular clusters and projections parallel to the axes.

Projective clustering There are some algorithms which can find clusters which are dense in a projection of the original data space. Proclus [APW+99] and DOC [PJAM02] search the space of axes-parallel projections to find good clusterings of the data. More advanced techniques like Orclus [AY00] and projective k-means [AM04] analyse eigenvalues of subsets of the data and can find arbitrary linear projections, in which points are clustered.

Fractals-related work Concepts of intrinsic dimensionality from fractals have been successfully used in the database field for numerous problems, such as nearest-neighbour queries [PKF00] and spatial query selectivity estimation [FK94, BF98]. Recent results [KL04] discuss doubling dimension as measure for intrinsic dimensionality. The proposed algorithm works efficiently, when the intrinsic dimensionality is bounded.

Barbará et al. [BC00] propose a clustering approach that uses the fractal dimension. However, it relies on the *global* dimension of the dataset, which tends to be dominated by the largest or most dense subsets of the data. In particular, for each point x_i, it estimates the global fractal (box-counting) dimension of the datasets X and of $X \setminus x_i$ and uses their difference to cluster the points.

Finally, LOCI (see Chapter 3) is an outlier detection method based on the local distribution of pairwise distances at multiple scales. Although the key concepts are loosely related, LOCI focuses on an entirely different application and does not use the concept of intrinsic dimensionality in any way.

5.2 Overview of the approach

In this section we give an overview of our method. As we discussed before, the method draws upon and extends previous density-based algorithms, as well as concepts of intrinsic dimensionality. The two key measures it uses are *local density* and *local dimensionality*. Both are obtained by fitting a line on a subset of points of the *local growth curve*.

First, we review basic facts about the notion of intrinsic dimension. Then, we describe local growth curves, and we explain how local density and local dimensionality are computed, and how they are used for clustering the dataset.

5.2.1 Background on intrinsic dimension

As an underlying basis of our method, we use the notion of *correlation dimension*, which is a measure of the *intrinsic dimensionality* of a dataset. In the following discussion, we assume

that the dataset X is a subset of $\mathbb{R}^m$, and for definition purposes we assume that the number of points n in X approaches the infinity. Let $d : X \times X \to \mathbb{R}$ be a distance function between pairs of points of X, and let $C(r)$ be the average number of pairs of points within distance r, that is,

$$C(r) = \lim_{n\to\infty} \frac{1}{n^2} \sum_{x\in X} |B(x,r)|,$$

where $B(x,r) = \{y \mid y \in X, d(x,y) \leq r\}$ is the subset of points contained in a ball of radius r, centred at point x. The correlation dimension is then defined as

$$d_{corr} = \lim_{r,r'\to 0} \frac{\log[C(r)/C(r')]}{\log[r/r']}. \tag{5.1}$$

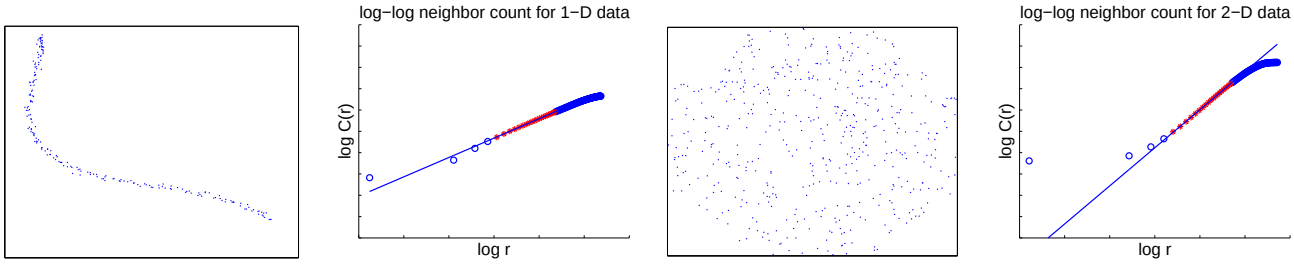
We assume that all the limits exist. Alternative definitions of intrinsic dimensionality can be found in the literature, such as *capacity* or *box counting dimension* and *information dimension*. Intrinsic dimensionality measures are sometimes also collectively referred to as *fractal dimension*. The interested reader can find a comprehensive development of the topic in a standard textbook, e.g., Rasband [Ras90].

In practice, we deal with finite sets, so the definition of correlation dimension in Equation (5.1) is not applicable. In this case, we define the function $C(r)$ as

$$C(r) = \frac{1}{n^2} \sum_{x\in X} |B(x,r)|, \tag{5.2}$$

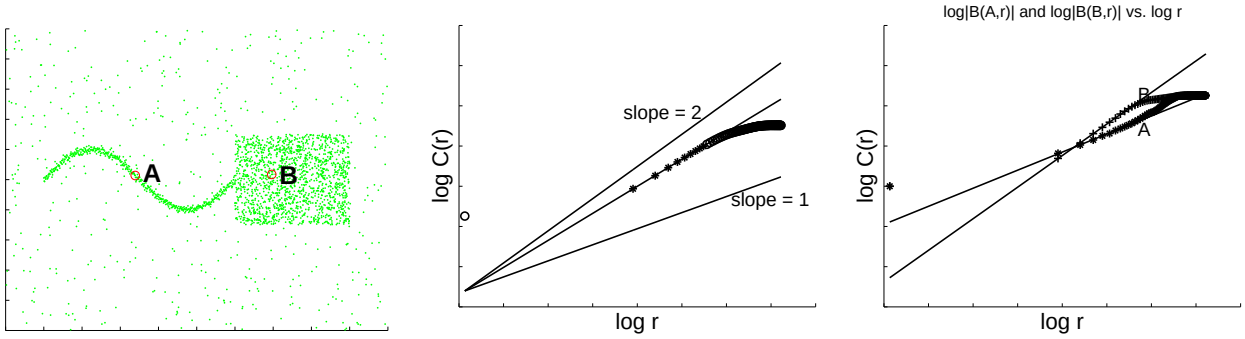
and estimate the correlation dimension by the *slope* of the function $C(r)$ in the log-log scale. The reason is that, since $\frac{\log[C(r)/C(r')]}{\log[r/r']} = \frac{\log[C(r)]-\log[C(r')]}{\log r - \log r'}$, the correlation dimension expresses the increase rate of $\log[C(r)]$ between $\log r$ and $\log r'$.

The intuition is shown in Figure 5.2: For points that are one-dimensionally arranged, as shown in Figure 5.2(a), one expects to find twice the number of points when doubling the radius. On the other hand, for points that are scattered on the 2-D plane, as shown in Figure 5.2(c), when doubling the radius, we expect the number of points to increase quadratically. The growth rates of the number of points in Figures 5.2(a) and 5.2(c) can then be estimated from the slope of the $C(r)$ curve in log-log scale, as shown in Figures 5.2(b) and 5.2(d), respectively. These are close to one and two, respectively. To enable visual comparison, the scales of the Figures 5.2(b) and Figures 5.2(d) are the same.



(a) (b) (c) (d)

Figure 5.2: Intuition behind the intrinsic dimensionality (correlation dimension).



(a) (b) (c)

Figure 5.3: A dataset that contains two subsets of different intrinsic dimensionality

5.2.2 Local Correlation Dimension

The function $C(r)$ as defined in Equation (5.2) computes the average number of neighbours of a point within distance r, where the average is taken over *all* points of the dataset. However, due to averaging, if the dataset is non-homogeneous, the estimated correlation dimension will not reflect the "true" dimensionality of the data. Figure 5.3 illustrates this point. Figure 5.3(a) shows a dataset with two distinct subsets of points: in the first subset the points lie on a 1-D curve, while the second subset consists of a cloud of 2-D points. As a consequence of taking averages, the intrinsic dimension of the whole dataset is somewhere between one and two. Figure 5.3(b) shows the line fitted to the $C(r)$ curve and, for comparison, lines with slopes one and two.

Therefore, in the case that a dataset consists of subsets with different intrinsic dimensionality, the correlation dimension of a dataset does not correctly characterise the dimensionality of the dataset. To overcome this problem we extend the definition of correlation dimension for each point in the dataset.

Definition 12 (Local-Growth Curve). *For each point x, we define the* local-growth curve, *to be*

the function of r, $G_x : \mathbb{R} \to \mathbb{N}$ *that computes the fraction of neighbours of x in a ball of radius r,*

$$G_x(r) = \lim_{n\to\infty} \frac{1}{n}|B(x,r)|.$$

The local growth curve G_x describes the density of the local neighbourhood of x for all distances r. In addition, the G_x curve contains information about the growth rate of the number of neighbours of x. We can now define the *local-correlation dimension* (or local dimension) of point x.

Definition 13 (Local-Correlation Dimension). *We define the* local-correlation dimension d_x *of point x, as*

$$d_x = \lim_{r,r'\to 0} \frac{\log[G_x(r)/G_x(r')]}{\log[r/r']}. \tag{5.3}$$

As in the case of correlation dimension, when dealing with finite sets, we define the local growth curve to be $G_x(r) = \frac{1}{n}|B(x,r)|$, and we compute the local-correlation dimension d_x of a point x by the slope of G_x curve in log-log scale. Notice that for finite sets the G_x curve is step-wise – its value chance only when the radius grows to include the next neighbour of a point. As a result, the local-growth curve can be losslessly represented by specifying its value on a finite set of radii $\mathcal{D}_x \subset \mathbb{R}$, which we call the *domain* of G_x.

5.2.3 Local representation

We now describe how to use the local growth curves and the local-correlation dimension in order to represent the dataset.

Let $X = \{x_1, \ldots, x_n\}$ be a dataset of n points in some metric space. For each point x_i we define its domain $\mathcal{D}_{x_i}$, and we compute the local growth curve G_{x_i}. We then take the G_{x_i} curve in log-log scale, and we find the line that fits it best, in a least squares sense. We use L_{x_i} to denote this line, and we call it the *linear growth model* for point x_i.

Definition 14 (Linear Growth Model). *We refer to the line*

$$L_{x_i}(\log r) = d_i \log r + b_i$$

as the Linear Growth Model *for the point* x_i.

The slope d_i of the line L_{x_i} is an estimate of the local-correlation dimension of point x_i. Using the linear growth model, we can now represent the point x_i using just two numbers.

The first number is the value d_i, the *local dimension* of the point x_i. The second number is denoted by $c_i = L_{x_i}(\log r^*)$ and it corresponds to the density of the dataset in a ball of radius r^*, centred on x_i *as it is estimated* by the linear growth model L_{x_i} for point x_i. We defer the discussion about the value r^* to Section 5.3, Lemma 5. We call c_i the *local density* of the point x_i. We write $l(x_i) = (d_i, c_i)$ to denote the representation of x_i by these two parameters.

Definition 15 (Local representation). *The mapping $l(x_i) = (d_i, c_i)$ is called the* local representation *of x_i, where $c_i = L_{x_i}(\log r^*)$.*

To illustrate the intuition behind local representation, consider two specific points A and B that come from the two different subsets in the example of Figure 5.3. Panel 5.3(c) shows $G_x(r)$ for $x = A$ and $x = B$ together with the fitted lines. In our example, the local dimensionality of point A is 1.20 and that of point B is 1.98. Therefore, we are able to distinguish the subsets with different intrinsic dimensionality using the local dimensionality d_i.

As we will explain in the next section, it is meaningful to ignore the parts of the local growth curve that correspond to very small and very large radii. The reason is that, for small r, the value of $G_x(r)$ is sensitive to local noise effects. Thus, ignoring small radii improves the robustness of the estimated dimension. On the other hand, for large r too many points contribute to the value of $G_x(r)$. Therefore, G_x does not capture local-neighbourhood structure around x any more; most curves look identical. For these reasons we restrict the domain $\mathcal{D}_x$ of $G_x(r)$ to a smaller subset $\mathcal{F}_x \subseteq \mathcal{D}_x$, which we call the *fitting set* of $G_x(r)$, and it is precisely the range over which we fit the linear-growth model L_x The details of how the fitting set is determined are discussed in Section 5.3.2.

5.2.4 Overall clustering

The final step of our method is to detect clusters of points that form low-dimensional manifolds in the ambient space of the dataset. Taking advantage of the simplicity of the local representation $l(x_i) = (d_i, c_i)$, we can perform this step using a standard clustering algorithm. Assuming that the local representation maintains well the information about the local density and the local dimensionality of the points, the clustering process is relatively easy since it is an operation on two-dimensional data (i.e., the pair of numbers forming the local representation). For our experiments we used the standard EM algorithm with full covariance matrices. Furthermore, the 2-D representation offers an informative visualisation of the dataset. In an user-interactive system it is usually easy to determine the underlying clusters in the dataset.

Finally, we note that the output of our algorithm can be further processed to discover dimensions of interest. First, in case that the algorithm places two manifolds of same dimension and density into one cluster, but the two manifolds do not intersect, then they can be separated by the single-linkage clustering. Second, in case of axis aligned subspaces, we can easily discover the attributes of interest by measuring the variance along each dimension. Finally, in case that we are interested in linear subspaces, running PCA will reveal the directions of interest. All of these three tasks become significantly easier once the appropriate subset of points has been identified by our method.

5.3 Analysis of our method

In this section we discuss the properties of our definitions, and their implications in the overall approach. As our guide in this discussion we consider the simple cases of points lying on a 2-D grid and on a 1-D line.

5.3.1 Discussion and Examples

The definition of correlation dimension in Section 5.2 assumes that the size of the set X is infinite. For an infinite real line and an infinite real plane the dimensions are precisely 1 and 2, respectively. In practice, however, we deal with finite sets with finite extent, so we can only compute an estimate of the actual dimension. We will now study the effect of finiteness on the local representation of the points by investigating two simple cases. The first dataset L consists of n one-dimensional points equally spaced on a line. The second dataset G consists of n two-dimensional points arranged on a grid. Ideally, the intrinsic dimensionality of the line should be one, and the dimensionality of the grid should be two. However, due to the finite size of the datasets, the estimated dimensionality is different.

Consider the points in the set L and assume that the point x_i is located at position i of the real line. Consider one of the endpoints of the line, e.g., the leftmost point x_1 of the line. The local growth curve of x_1 is $G_{x_1}(r) = \frac{r}{n}$ (when computing $|B(x_1, r)|$ we do not count the point itself). For the point x_m in the middle of the line $m = n/2$, the local growth curve is $G_{x_m}(r) = \frac{2r}{n}$. In both of these cases, the local dimensionality of points x_1 and x_m is one, as expected. However, consider the point x_p that lies in position $p = n/4$. For radii $r = 1..\frac{n}{4}$, $G_{x_p}(r) = \frac{2r}{n}$, while for radii $r = \frac{n}{4}...\frac{3n}{4}$, $G_{x_p}(r) = \frac{r}{n}$. Due to this change in the local growth curve, when fitting a line the local dimensionality of the point x_p is underestimated. For example, for a line with 500 points, the local dimensionality is estimated to be around 0.87.

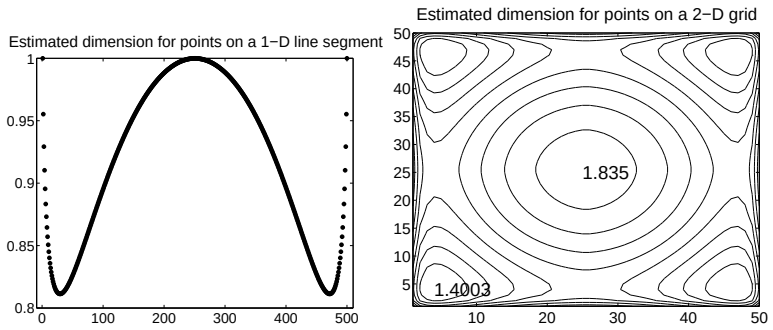


Figure 5.4: Boundary effects on the estimation of the correlation dimension.

The d_i values for all points are shown in Figure 5.4.

Determining the correct slope is even harder for the two-dimensional grid. Consider the point x_m in the middle of the grid. For simplicity we will assume that the distance between points is measured using the L_∞ norm. It is not hard to see that the local growth curve for this point is $G_{x_m}(r) = \frac{1}{n}((2r+1)^2 - 1) = \frac{1}{n}(4r^2 + 4r)$ (again, the point x_m is not counted in the computation). Due to the additive term $4r$ we need to have $r \to \infty$ in order for the local dimension d_m of x_m to tend to 2. In practice, this results in underestimating the dimension of the point. For a 50×50 grid the local dimension of the middle points is estimated to be close to 1.6. Furthermore, simple computations show that for a point x_s on the side of the grid, $G_{x_s}(r) = \frac{1}{n}(2r^2 + 3r)$, while for a point x_c on the corner of the grid, $G_{x_s}(r) = \frac{1}{n}(2r^2 + 3r)$. Again, the local growth curve on the boundary of the grid is different from that inside the grid. Therefore, when the curve hits the boundary there is a change in the local growth curve, which results in further underestimation of the local dimension. An example with a 50×50 grid is shown in Figure 5.4. The contour lines show how the local dimension changes for different points of the grid. The maximum and minimum values are shown on the plot.

We next consider the case of a dataset consisting of a line embedded in a grid. We assume that the line consists of grid points which are replicated μ times. The value μ is the density of the line. For simplicity, assume that the line lies in the middle of the grid and it is parallel to one axis of the grid. Furthermore, in order to avoid dealing with boundary points, assume that the grid extends to infinity in all directions. For a point on the line x_ℓ it is not hard to show that the local growth curve is $G_{x_m}(r) = \frac{1}{n}((2r+1)^2 + \mu(2r+1) - 1)$. When the value μ is large enough compared to r the growth of $G_{x_m}(r)$ is dominated by the linear term. Of course as $r \to \infty$, the quadratic part becomes dominant. For a grid point, the growth is the same as before as long as the ball around the point has not reached the line. When the line is reached, the local growth curve becomes the same as for a line point (this is also due to the

fact that we consider the L_∞ distance, and the line is parallel to the axis).

In the next paragraph we will see how to address the issues raised by the previous examples. The idea is to compute the local dimensionality d_x of each point by fitting a line not to the entire local growth curve G_x, but only on the fitting set $\mathcal{F}_x$. We discuss how we determine $\mathcal{F}_x$ next.

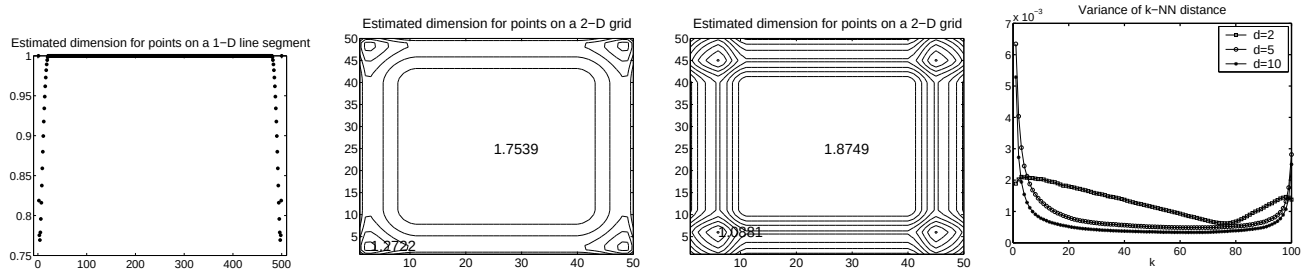
5.3.2 Determining the fitting set

An important issue in the definition of local growth curves is the domain of radii over which they are defined. An immediate idea is to define the curves over the interval ranging from the minimal pairwise distance up to the diameter of the dataset. Let $\mathcal{R}$ denote this interval. This is the maximal interval over which the local growth curves can be defined. However, this approach is extreme, since for most points, the low part of the curve will be zero (balls with small radius contain no points), while the upper part of the curve will be one (balls with large radius contain the whole dataset). This will result in poor estimates for the local dimension of these points.

One approach for dealing with this problem is to restrict the definition of the local growth curves over an interval $[r_{\min}, r_{\max}] \subset \mathcal{R}$, which one might believe that captures the useful information of the local growth curve. However, this approach is also problematic when the density of the dataset differs in different regions of the space. Furthermore, for points that lie in large dimensional spaces, the minimum and maximum distances converge, so it is challenging to find a meaningful interval.

To address these issues, we choose to define a different domain $\mathcal{D}_x$ for each point in the dataset. This domain is defined by growing a ball around x such that at each step we extend the ball to include (at least) one more neighbour. In other words, the domain $\mathcal{D}_x$ is precisely the set of radii $\{r_1, r_2, ..., r_n\}$, where r_k is the distance of x to its k-th nearest neighbour.

Following the discussion in Section 5.3.1, it becomes clear that it is beneficial to restrict the domain $\mathcal{D}_x$ by considering the distances only up to some k_{max}-th nearest neighbour, instead of all possible neighbours. This has the following advantages. First, it captures best the idea of locality upon which our approach is based. As it was demonstrated in the case of the line embedded in the grid, this can help discriminate between points that lie on different manifolds. Second, it helps in avoiding strong boundary effects, since fewer points hit the boundaries, and thus we can better estimate their "real" dimension. This is shown in Figures 5.5 *(a)*, *(b)*, and *(c)*, where by restricting the interval from above, we obtain a better estimation of the dimension for more points of the line and the grid.



(a) (b) (c) (d)

Figure 5.5: Restricting the fitting interval

We further restrict the interval R_x from below, by considering only the neighbours that are no closer than the $k_{\min}$-th nearest neighbour. As discussed in Section 5.3.1, in the case of the grid this helps obtain a better estimate of the dimension. This becomes obvious when comparing the the figures *(b)* and *(c)* in Figure 5.5. Figure *(c)* is obtained by restricting the interval R_x from below, where we obtain an estimate of the dimension closer to 2.

Furthermore, for small values of k (i.e., for the very first nearest neighbours) the radius r_k might be affected by small local variations of the density of the points. Such density variations might include isolated points or unusually dense areas. Our point is illustrated in Figure 5.5*(d)* 100 points have been generated uniformly at random in a d-dimensional hypercube, for $d = 2$, 5, and 10. By repeating the process of random point generation 1000 times, we estimate the expected distance and the variance of the k-th nearest neighbour from a randomly selected point as a function of k. Figure 5.5*(d)* shows that the variance of the distances of the very first nearest neighbours is large. Therefore, by ignoring the k-th nearest neighbours for $k < k_{\min}$ we obtain a more robust estimation of the local dimension. We are now ready to summarise with the following simple definition.

Definition 16 (Fitting set). *The set of radii* $\mathcal{F}_x = \{r_k \in \mathcal{D}_x \mid r^{(x)}_{k_{\min}} \leq r_k \leq r^{(x)}_{k_{\max}}\}$, *where* $r^{(x)}_{k_{\min}}$ *and* $r^{(x)}_{k_{\max}}$ *are the distances of the* $k_{\min}$*-th and* $k_{\max}$*-th nearest neighbours of* x *is called the* fitting set.

In our experiments, we have found that the algorithm is not particularly sensitive in the choice of $k_{\min}$ and $k_{\max}$. For example, the values $k_{\min} = 0.01 \cdot n$ and $k_{\max} = 0.1 \cdot n$ give good quality of results in a wide variety of datasets.

5.3.3 Estimating local density

In this paragraph we derive an estimation for the radius r^*, which is used for computing the local density $c_i = L_{x_i}(\log r^*)$ for each x_i. For simplicity of notation we rewrite Equation (14)

as $Y = d_i X + b_i$. We also write $X^* = \log r^*$ and $Y^* = d_i X^* + b_i = c_i$. Notice that by fitting a line to the curve G_{x_i} we obtain the parameters d_i and b_i. The goal is to compute the "best" choice of parameter $\log r^*$ that achieves the local representation $l(x_i) = (d_i, c_i)$ for each x_i.

The main observation is that by setting $\log r^* = +\infty$ in Equation (14), the resulting values c_i of local densities are perfectly correlated with the values d_i of local dimensions. The reason is that for $\log r^* = +\infty$ the ordering of c_i's is completely determined by the ordering of d_i's. Similarly, for $\log r^* \to -\infty$, the c_i's are perfectly anti-correlated with d_i's. Since our goal is to use the pair (d_i, c_i) that capture as much information for each x_i as possible, we would like to choose r^* so that the parameters d_i and c_i are uncorrelated. Based on this idea we can estimate the optimal radius r^* for the local representation $l(x_i)$. Note that it makes a huge difference for the visualisation as well as for automated clustering algorithms, whether the two-dimensional data (d_i, c_i) are uncorrelated or not.

Lemma 5. *The choice of r^* for which d_i and c_i are uncorrelated is given by*

$$\log r^* = -\frac{\sum(d_i - \bar{d})(b_i - \bar{b})}{\sum(d_i - \bar{d})^2}$$

Proof. The correlation between the variables d_i and c_i can be estimated by the coefficient

$$r_{dc} = \frac{\sum_i (d_i - \bar{d})(c_i - \bar{c})}{\sqrt{\sum_i (d_i - \bar{d})^2 \sum_i (c_i - \bar{c})^2}},$$

where $\bar{d} = E[d_i]$ and $\bar{c} = E[c_i]$ are the expectations of d_i's and c_i's, respectively. To make the correlation zero we need to choose r^* so that the numerator of r_{dc} is equal to zero. Let $\bar{b} = E[b_i]$ be the expectation of b_i's. Since $c_i = d_i X^* + d_i$, by linearity of expectation we get $\bar{c} = E[c_i] = E[b_i + X^* d_i] = \bar{b} + X^* \bar{d}$. The numerator of the correlation coefficient can now be written as

$$\begin{aligned}
&\sum(d_i - \bar{d})(c_i - \bar{c}) \\
&\quad = \sum(d_i - \bar{d})(b_i + X^* d_i - \bar{b} - X^* \bar{d}) \\
&\quad = \sum(d_i - \bar{d})(X^*(d_i - \bar{d}) + (b_i - \bar{b})) \\
&\quad = X^* \sum(d_i - \bar{d})^2 + \sum(d_i - \bar{d})(b_i - \bar{b}).
\end{aligned}$$

Setting $r_{dc} = 0$ gives the optimal value of $\log r^*$.

Input: Dataset X of n points, number of clusters b
Ouput: Clustering of X into b clusters
1: **for all** $i \in \{1, \ldots, n\}$ **do**
2: Compute k-th NN of x_i, for $k = k_{\min} \ldots k_{\max}$
3: Compute the local representation (d_i, c_i) of x_i.
4: **end for**
5: $X_{LR} = \{(d_1, c_1), \ldots, (d_n, c_n)\}$
6: Cluster the set X_{LR} into b clusters.

Figure 5.6: The DIC algorithm

5.4 The algorithm

We now present the Dimension Induced Clustering (DIC) algorithm. The algorithm works on the representation of the dataset defined in Section 5.2. The objective of the algorithm is to partition points so that points in the same cluster lie on dense manifolds of the same dimension.

5.4.1 DIC algorithm

The outline of the DIC algorithm is shown in Figure 5.6. The input to the algorithm is a set X of n elements, that we want to cluster in b clusters. In first step, the algorithm computes for each element x_i, the distance of x_i to its k-th nearest neighbour for all $k = k_{\min}, \ldots, k_{\max}$. The distances of the nearest neighbours of x_i specify completely the local growth curve G_{x_i}. By fitting the linear growth model L_{x_i} on G_{x_i} and by estimating the local density, as in section 5.3.3, we compute the local representation $l(x_i) = (d_i, c_i)$ for each x_i. Thus, we map the set X into a two dimensional set X_{LR} that contains the local representation of all points. The task now becomes to cluster the two-dimensional points in X_{LR}. Clustering in two dimensions is conceptually much simpler than clustering in high-dimensional spaces. The correct clustering can often be determined even by simple visual inspection. In the automated case applying a EM (Expectation Maximisation) algorithm [HTF01b] for fitting b Gaussian distributions on the data works well in many cases. If the set X consists of b sufficiently dense subsets that lie on manifolds of different dimension, which are sufficiently separated, the algorithm will be able to separate these subsets.

5.4.2 Efficiency of DIC

The complexity of the DIC algorithm is dominated by the complexity of computing for every point x_i the distance to the k_{min} to $k_{\max}$ neighbours of the point x_i. The simple solution to this problem is to compute the distances between all points in the set X, and for each point x_i sort the points with respect to their distance from point x_i, and retrieve the necessary information. The time for computing all pairwise distances is $O(n^2)$.

A different approach is to construct an index for the elements in X that supports fast execution of k-nearest neighbour queries. In case that X consists of vector data, spatial index structures can be used for the efficient calculation such as [AMN+98, KS97, BKK96]. In case of metric data the OMNI framework [FTCTF01], or data structures like the M-tree [CPZ97] can be used. Since the computation of the local representation is inherently approximate, the use of approximative methods for k-nearest neighbour queries such as locality-sensitive hashing [GIM99], is also possible.

Investigating the construction of the appropriate nearest neighbour index is beyond our scope. We assume that such an index exists, and we use it as a black box for obtaining the distances of the k-th nearest-neighbour queries for $k = k_{\min}, \ldots, k_{\max}$. The efficiency of the DIC algorithm is determined by the efficiency of this index.

5.5 Experiments

In this section we study experimentally the properties and the performance of the DIC algorithm.

5.5.1 Applications and Datasets

We apply our algorithms on the following types of datasets.

Embedded m-flats: Consider a set X of n points in $\mathbb{R}^d$ that can be decomposed in two subsets N, and F, of size s and f respectively, where $n = s + f$. The points in N are are distributed uniformly at random in $(0,1)^d$. For the points in F, in the first $d - m$ coordinates they take values normally distributed around 0.5, with variance 0.01. In the last m coordinates they take values uniformly distributed in $(0,1)$. As $s, f \to \infty$ the intrinsic dimensionality of the sets N and F approaches d and m respectively. We call the set F an *m-flat*. The value m is the dimension of the m-flat. The set N can be thought of as an m-flat of dimension d, so we say that N has *full dimension*.

The objective of the algorithms is to partition the set X into sets N and F. We apply the DIC algorithm on X, requesting 2 clusters. We will demonstrate that the DIC algorithm, is able to return the sets F and N as the clusters even when m and d are relatively close. The flat F is the set of nodes with the smaller average intrinsic dimensionality.

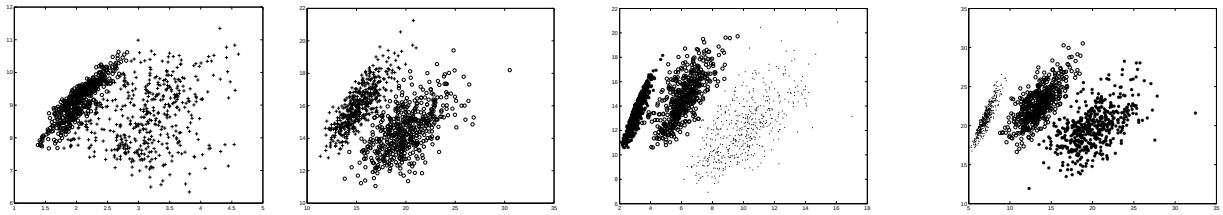
Manifolds within manifolds: The setting is similar to the previous one, only this time the set X contains more than one m-flats of different dimensions. Namely, the set X can be decomposed into sets $N, F_1, \ldots, F_p$, where N has full dimension d, and $F_1, F_2, \ldots, F_p$ are m-flats with dimensions $m_1 < m_2 < \ldots < m_p$ respectively. The m-flats are constructed as described above. Note that since for every flat F_i we always "fix" the first $d - m_i$ coordinates, the m-flats with lower dimension are embedded within the m-flats of higher dimensions. This results in creating a chain hierarchy of manifolds where every manifold is embedded in all the preceding ones in the chain.

Again, we apply the DIC algorithm, requesting $p + 1$ clusters. When the dimensionalities of the m-flats are sufficiently separated, the algorithm returns as clusters that p flats and the set N. The average estimated dimension for each set are ordered according to the actual dimension of the flats.

Low Rank Submatrices: The input is an $n \times m$ matrix that takes values in $[0, 1]$. Within the matrix there is a collection of k rows and ℓ columns, such that the combinatorial $k \times \ell$ submatrix has low rank. The objective is to identify the rows and columns of this submatrix.

We generate such datasets as follows. First we generate a $k \times \ell$ matrix S of rank exactly r, where $r \ll \min\{n, m\}$. We then plant it in the matrix M. The remaining elements of M are generated uniformly at random, scaled so that the mean is zero and the standard deviation is one. Therefore, if we remove either the k rows, or the ℓ columns of matrix S from M, we obtain a matrix of rank $\min\{n - k, m\}$ and $\min\{n, m - \ell\}$ respectively. To this matrix we add a "noise" matrix X with entries distributed normally around 0, with variance 0.05.

In order to extract S from the matrix M we apply the DIC algorithm in two steps. First we perform a clustering of the rows, and we identify the rows of the matrix S. The dimension of these rows is $m - \ell$, as opposed to m which is for the rest of the rows, so it is easy for the DIC algorithm to identify them. We then cluster the columns of M. The dimension of the columns in S is $n - k$ as opposed to n for the rest of the columns, so again DIC manages to partition the rows. Given the rows and columns we can extract matrix S.



(a) 2D-flat in 3D (b) 40D flat in 50D (c) 3D and 6D flats in 10D (d) 10D and 20D flats in 30D

Figure 5.7: Discovering m-flats with DIC

5.5.2 Experiments with DIC

In this section we present experiments with DIC algorithm on various datasets. In all runs of the algorithm, we set $k_{\min} = 10$, and $k_{\max} = 100$, two values that we observed that they work well in practice.

We start by experimenting with datasets that contain a single m-flat F, embedded in a space of higher dimension d, together with a set N of noise points distributed uniformly at random. The datasets are constructed as described in section 5.5.1. Since the objective is to separate the set F and N, we evaluate our algorithm by looking into the *total classification error* of the algorithm. The total classification error E_{tot} is computed as follows: We first compute the confusion matrix C whose C_{ij} entry contains the number of overlapping points between the i-th cluster of the ground truth and the j-th cluster of the clustering found by the algorithm. Then $E_{tot} = 1 - (\sum_i \max_j C_{ij})/n$.

Our experiments indicate that the DIC algorithm performs exceptionally well in this setting, even in the case that the dimension of the host space and the m-flat are very close, or if the m-flat is embedded in a high dimensional space. Figure 5.7(a) plots the local representation of the data points when $d = 3$ and $m = 2$, and their clustering. Figure 5.7(b) shows the the case where $d = 50$ and $m = 40$. In both cases, the size of the dataset is 1,000 points, of which 500 belong to the m-flat. We observe that the algorithm manages to identify the m-flats successfully. The total classification error is 8.1% in the first case, and 1.2% in the second case.

In order to better understand the performance of DIC, we performed a more detailed experiment, generating datasets with the dimension of the host space being $d = 2 \ldots 10$, and the dimension of the m-flat ranging from 1 to $d - 1$. In all cases, the dataset consists of 1,500 points, 500 of which belong to the m-flat. Table 5.1 reports the average classification error for 20 runs of the algorithm (the numbers are percentages). We observed that the classification error is never more than 39%, and this occurs in the case that the dimension of the host space and that of the m-flat differ by just one.

	1	2	3	4	5	6	7	8	9
2	9.2								
3	13.0	20.14							
4	14.9	1.53	29.28						
5	16.1	0.26	6.74	26.42					
6	15.4	0.08	0.68	6.99	31.1				
7	7.1	0.02	0.17	1.25	13.7	33.4			
8	10.5	0.00	0.02	0.41	2.1	14.6	36.3		
9	1.4	0	0.01	0.08	0.6	2.9	18.7	37.9	
10	7.4	0.01	0.01	0.04	0.2	0.9	4.2	20.7	38.3

Table 5.1: Classification error of discovering m-flat clusters

We now turn our attention to cases where there are more than one m-flats in the dataset. Figures 5.7(c) and (d) show the plots of the local representations of two datasets that contain flats of different dimension. In the first case the dimension of the host space is $d = 10$ and the two manifolds have dimension $m_1 = 3$, and $m_2 = 6$. In the second case we have $(m_1, m_2, d) = (10, 20, 30)$, In both cases all three sets of points F_1, F_2, N contain 500 points each. We observe that the DIC algorithm manages to discriminate the three sets. The average classification error is 1.53% for the first case, and 0.51% for the second case, where the average is taken over 20 runs. Datasets with more than three flats are examined in the Appendix.

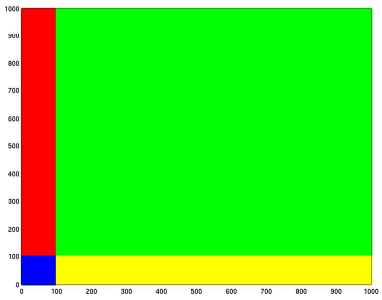


Figure 5.8: Discovering low rank matrices

We also experiment with low rank matrices, trying to detect a (combinatorial) 100×100 submatrix of rank 2, within a 1000×1000 matrix. The algorithm proves to be quite successful, obtaining classification error just 0.16%, where the average is taken over 10 runs. In this case the large dimension of the matrix works in favour of our algorithm. The algorithm often achieves a perfect partition of the matrix (4 out of the 10 runs). A case where we obtain a perfect partition of the matrix is shown in Figure 5.8. The blue part of the figure shows the correctly found low rank submatrix, the red and yellow areas correspond to the attached random columns and rows and the green part totally belongs to the random matrix.

The problem of finding low rank submatrices has been applied to microarray data. Wang

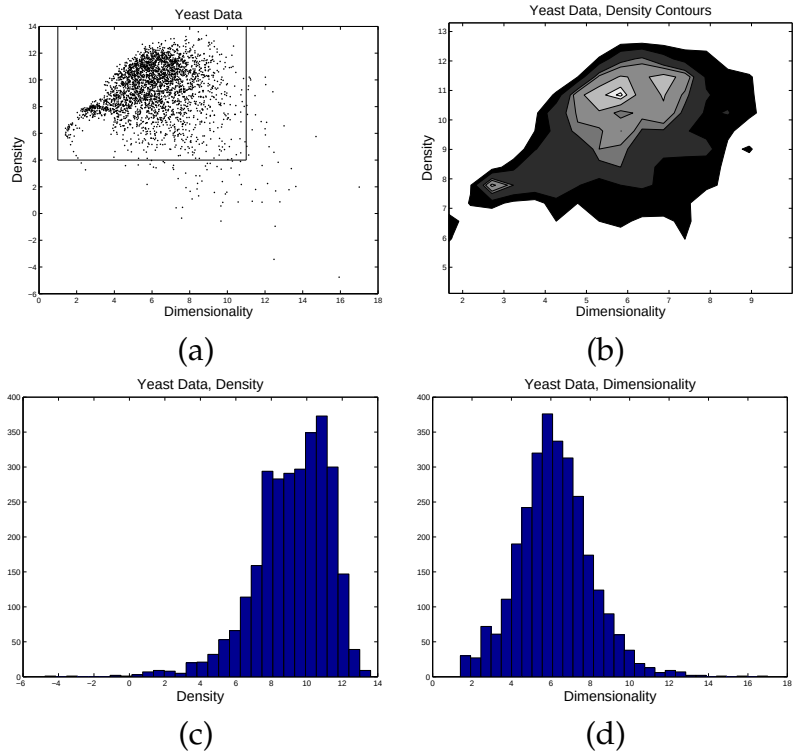


Figure 5.9: Analysis of microarray data, (a) dimensionality and density for each row, (b) density contours in the rectangular area of (a) reveal two clusters, (c) 1D histogram of the density component, (d) 1D histogram of the dimensionality component.

et. al. [WWYY02] report a solution for rank-1 submatrices. We experiment with the same microarray data used in [WWYY02][1]. The data contains the expression levels of 2884 genes (rows) for 17 different patients (columns). Finding combinatorial low-rank matrices in such types of data is important since they represent subsets of genes that are co-regulated on some subsets of patients. Figure 5.5.2a shows a plot of the local representation of the rows (genes) of the matrix. The scatter plot shows no obvious clustering except a few outliers. A more detailed density estimation of the 2D data reveals two hidden clusters. The figure 5.5.2b shows the density contour lines, estimated by a 30×30 histogram (light grey means high density). There is one small cluster the rows of which has dimensionality around 3 and lower density than the bigger cluster with intrinsic dimensionality of about 6. Note that the two clusters can not be found using either density or intrinsic dimensionality, as shown by the two 1D histograms of the both measurements (fig. 5.5.2c,d). This shows that our idea of combining density with intrinsic dimensionality is a real step forward.

[1]The data can be obtained at http://arep.med.harvard.edu/biclustering/yeast.matrix

5.6 Conclusions

We address the problem of discovering clusters of points that lie on low-dimensional manifolds. Our approach is to extend the definition of fractal correlation dimension and create a local-growth model for each point. Based on this model, each point in the dataset can be mapped to a local representation consisting of a density coefficient and a dimensionality coefficient. We argue that the local representation maintains well the information about the manifolds that points belong to, and discovering those manifolds becomes a two-dimensional clustering problem.

Our method is able to discover low-dimensional manifolds that are not necessarily linear, it can find clusters within clusters, as well as clusters that occupy the same space. Furthermore the method does not require a vector-space representation of the data; it can be used equally well for metric datasets. We perform experiments in which we demonstrate the effectiveness of our algorithms for discovering low-dimensional m-flats and for detecting low-rank sub-matrices. We also show that our method outperforms other approaches that are based only on density and do not take into account the notion of dimensionality.

Chapter 6

Summary

In this part we considered homogeneous spatial data, where each point has only spatial features. Our techniques rely on examination of the dataset at multiple distance scales. We employ both the first and the second moments of pairwise distance distributions.

In Chapter 3 we presented a new method for evaluating "outlier-ness," which we call the *Local Correlation Integral* (*LOCI*). As with the best previous methods, LOCI is highly effective for detecting outliers and groups of outliers (*a.k.a.* micro-clusters). In addition, it offers the following advantages and novelties: *(a)* It provides an automatic, data-dictated cut-off to determine whether a point is an outlier—in contrast, previous methods force users to pick cut-offs, without any hints as to what cut-off value is best for a given dataset. *(b)* It can provide a LOCI plot for each point; this plot summarises a wealth of information about the data in the vicinity of the point, determining clusters, micro-clusters, their diameters and their inter-cluster distances. None of the existing outlier-detection methods can match this feature, because they output only a single number for each point: its outlier-ness score. *(c)* Our LOCI method can be computed as quickly as the best previous methods. *(d)* Moreover, LOCI leads to a practically linear approximate method, *aLOCI* (for *approximate LOCI*), which provides fast highly-accurate outlier detection. To the best of our knowledge, this is the first work to use approximate computations to speed up outlier detection.

In Chapter 4 we presented *outliers by example (OBE)* [ZKPF04] to incorporate user feedback, without exposing users to any parameters. Instead, we try to infer the (few) parameters from user examples. A fundamental issue is that the notion of which objects are outliers typically varies between users or, even, datasets. We present a novel solution to this problem, by bringing users into the loop. Our OBE (Outlier By Example) system is, to the best of our knowledge, the first that allows users to give some examples of what they consider as outliers. Then, it can directly incorporate a small number of such examples to successfully

discover the hidden concept and spot further objects that exhibit the same "outlier-ness" as the examples.

In Chapter 5 we presented *dimension induced clustering (DIC)* [GHPT05] and develop a parameter-free transformation of high-dimensional points into a pair of local dimension and local density features, which can be used to cluster points belonging to low-dimensional manifolds. It is commonly assumed that high-dimensional datasets contain points most of which are located in low-dimensional manifolds. Detection of low-dimensional clusters is an extremely useful task for performing operations such as clustering and classification, nevertheless, it is a very challenging computational problem. We study the problem of finding subsets of points with low intrinsic dimensionality. Our main contribution is to extend the definition of fractal correlation dimension, which measures average volume growth rate, in order to estimate the intrinsic dimensionality of the data in local neighbourhoods. We provide a careful analysis of several key examples in order to demonstrate the properties of our measure. Based on our proposed measure, we introduce a novel approach to discover clusters with low dimensionality. The resulting algorithms extend previous density based measures, which have been successfully used for clustering.

Part II

Spatial mining — Heterogeneous

Chapter 7

Introduction

In several applications, each object in the dataset may have non-spatial attributes (or features), besides its spatial location. For example, galaxies may belong to one of two types (say, spiral or elliptical) or patches of land may contain several among tens or hundreds of different species (see also Section 1.4.1). In this part of the thesis we examine two possible combinations of spatial and non-spatial attributes.

In Chapter 8 we consider the case of a single non-spatial attribute which can take one of two possible values. Equivalently, each point in the dataset may belong to one of two classes. In this setting, we examine the problem of detecting *cross-outliers*, i.e., outliers in one class with respect to the other. The proposed approach extends the methods in Chapter 3 and examines the implications of having two classes of points. More specifically, to the best of our knowledge, work on outliers up to date focuses exclusively on the problem as follows [Haw80]: "given a *single* set of observations in some space, find those that deviate so as to arouse suspicion that they were generated by a different mechanism." Instead, we consider the problem "given a set of observations with class labels, find those that arouse suspicions, taking into account the class labels." A single observation may look normal both within its own class, as well as within the entire set of observations. However, when examined with respect to other classes, it may still arouse suspicions. Many of the existing outlier detection approaches cannot be extended to this case. We present one practical approach for dealing with this problem.

In Chapter 9 we allow an arbitrary number of binary non-spatial features, for each spatial location. We restrict the spatial locations to lie on a rectangular grid. In other words, we deal with raster data but with the value of each pixel being a multidimensional binary vector. In this setting, we examine with the problem of *simultaneously* finding spatial correlation patterns and feature co-occurrence patterns, without *any* parameters. In particular, we employ

the Minimum Description Length (MDL) principle coupled with a natural way of compressing regions. This defines what "good" means: a feature co-occurrence pattern is good, if it helps us better compress the set of locations for these features. Conversely, a spatial correlation is good, if it helps us better compress the set of features in the corresponding region. Our approach is scalable for large datasets (both number of locations and of features).

Chapter 8

Cross-outliers

As noted in [Haw80], "the intuitive definition of an outlier would be 'an observation that deviates so much from other observations as to arouse suspicions that it was generated by a different mechanism'." The traditional and—to the best of our knowledge—exclusive focus has been on the problem of detecting deviants in a single set of observations, i.e.,

Problem 1 (Outlier detection—single set). *Given a set of objects, find these that deviate significantly from the rest.*

However, there are several important practical situations where we have two collections of points. Consider the following illustrative example: Assume we have the locations of two types of objects, say vegetable patches and rabbit populations. If we consider, say, rabbit populations in isolation, these may be evenly distributed. The same may be true for food locations alone as well as for the union of the two sets.

Even though everything may look "normal" when we ignore object types, there is still the possibility of "suspicious" objects when we consider them in relation to objects of the other type. For example, a group of patches with far fewer rabbits present in the vicinity may indicate a measurement error. A population away from marked food locations may hint toward the presence of external, unaccounted-for factors.

The above may be considered a "toy" example that only serves illustrative purposes. Nonetheless, in several real-world situations, the spatial relationship among objects of two different types is of interest. A few examples:

- Situations similar to the one above actually do arise in biological/medical domains.
- In geographical/geopolitical applications, we may have points that represent populations, land and water features, regional boundaries, retail locations, police stations,

crime incidence and so on. It is not difficult to think of situations where the correlations between such different objects are important.

- In astrophysics, it is well known that the distributions of different celestial objects follow certain laws (for example, elliptical and exponential galaxies form small clusters of one type and these clusters "repel" each other). There are *vast* collections of astrophysical measurements and even single deviant observations would potentially be of great interest.

In brief, we argue that the following outlier detection problem is of practical importance:

Problem 2 (Cross-outlier detection). *Given two sets (or classes) of objects, find those which deviate with respect to the other set.*

In this case we have a primary set $\mathbb{P}$ (e.g., elliptical galaxies) in which we want to discover *cross-outliers* with respect to a reference set $\mathbb{R}$ (e.g., spiral galaxies). Note that the single set case is always a special case, where $\mathbb{R} = \mathbb{P}$.

However, the converse is *not* true. That is, approaches for the single-set problem are not immediately extensible to cross-outlier detection. First off, several outlier definitions themselves cannot be extended (see also Section 8.4.1), let alone the corresponding methods to apply the definitions and compute the outliers. In summary, the contributions in this chapter are two-fold:

- We identify the problem of cross-outlier detection. To the best of our knowledge, this has not been explicitly studied in the past, even though it is of significant practical interest. In general, an arbitrary method for the single-set problem cannot be easily extended to cross-outlier detection (but the opposite is true).

- We present a practical method that solves the problem. The main features of our method are:

  - It provides a meaningful answer to the question stated above, using a statistically intuitive criterion for outlier flagging (the local neighbourhood size differs more than three standard deviations from the local averages), with no magic cut-offs.

  - Our definitions lend themselves to fast, single-pass estimation using box-counting. The running time of these methods is typically linear with respect to both dataset size and dimensionality.

– It is an important first step (see also Section 8.4.3) toward the even more general problem of multiple-class cross-outliers (where the reference set $\mathbb{R}$ may be the union of more than one other class of objects).

The rest of this chapter is organised as follows: Section 8.1 briefly discusses related work for the single class case, as well as more remotely related work on multiple dataset correlations and clustering. Section 8.2 presents our definition of a cross-outlier and briefly discusses its advantages. Section 8.3 demonstrates our approach on both synthetic and real datasets. Section 8.4 discusses some important issues and possible future directions. Finally, Section 8.5 gives the conclusions.

8.1 Background and related work

In this section we present prior work on the problem of single class outlier detection. To the best of our knowledge, the multiple class problem has not been explicitly considered.

8.1.1 Single dataset outlier detection

Previous methods for single dataset outlier detection broadly fall into the following categories: distribution based, clustering based, depth based, distance based and density based. These were reviewed in Section 3.1.

8.1.2 Multiple class outlier detection

To the best of our knowledge, this problem has not received explicit consideration to this date. Some single class approaches may be modified to deal with multiple classes, but the task is non-trivial. The general problem is open and provides promising future research directions. In this section we discuss more remotely related work.

Multi-dimensional correlations The problem of discovering general correlations between two datasets has been studied to some extent, both in the context of data mining, as well as for the purposes of selectivity estimation of spatial queries. However, none of these approaches deal with single points and identification of outlying observations.

[TTPF01] deals with the problem the general relationship of one multi-dimensional dataset with respect to another. This might be a good first step when exploring correlations between

datasets. However, even when two datasets have been found to be correlated as a whole and to some extent co-located in space, this method cannot identify single outlying points.

Prior to that, [FSJT00] considers the problem of selectivity estimation of spatial joins across two point sets. Also, [BF95, BBKK97] consider the selectivity and performance of nearest neighbour queries within a single dataset.

Non-spatial clustering Scalable algorithms for extracting clusters from large collections of spatial data are presented in [NH94] and [KN96]. The authors also combine this with the extraction of characteristics based on non-spatial attributes by using both spatial dominant and non-spatial dominant approaches (depending on whether cluster discovery is performed first or on subsets derived using non-spatial attributes). It is not clear if these results can be extended to deal with the multiple class outlier detection problem. In the single class case, clusters of one or very few points can be immediately considered as outliers. However, this is not necessarily the case when dealing with multiple classes.

8.2 Proposed method

In this section we introduce our definition of an outlier and discuss its main properties. Our approach is based on the distribution of distances between points of the primary set and a reference set with respect to which we want to discover outliers. We use an intuitive, probabilistic criterion for automatic flagging of outliers.

8.2.1 Definitions

The definitions in this section are similar to those in Section 3.2.1, extended to the case of two classes of points. In particular, here we consider the problem of detecting outlying observations from a primary set of points $\mathbb{P}$, with respect to a reference set of points $\mathbb{R}$. We want to discover points $p \in \mathbb{P}$ that "arouse suspicions" with respect to points $r \in \mathbb{R}$. Note that single-set outliers are a special case, where $\mathbb{R} = \mathbb{P}$.

Table 8.1 describes all symbols and basic definitions. To be more precise, for a point $p \in \mathbb{P}$ let $\hat{n}_{\mathbb{P},\mathbb{R}}(p, r, \alpha)$ be the average, over all points $q \in \mathbb{P}$ in the r-neighbourhood of p, of $n_{\mathbb{R}}(q, \alpha r)$. The use of two radii serves to decouple the neighbour size radius αr from the radius r over which we are averaging.

We eventually need to *estimate* these quantities (see also Figure 8.1). We introduce the following two terms:

Symbol	Definition
$\mathbb{P}$ p_i	Primary set of points $\mathbb{P} = \{p_1, \ldots, p_i, \ldots, p_N\}$.
$\mathbb{R}$ r_i	Reference set of points $\mathbb{R} = \{r_1, \ldots, r_i, \ldots, r_M\}$.
N, M	Point set sizes.
k	Dimension of the data sets.
$d(p, q)$	Distance between points p and q.
$R_{\mathbb{P}}$, $R_{\mathbb{R}}$	Range (diameter) of each point set—e.g., $R_{\mathbb{P}} := \max_{p,q \in \mathbb{P}} d(p, q)$.
$\mathcal{N}_P(p, r)$	The set of r-neighbours of p from the point set P, i.e., $$\mathcal{N}(p, r) := \{q \in P \mid d(p, q) \leq r\}$$ Note that p does not necessarily belong to P.
$n_P(p, r)$	The number of r-neighbours of p_i from the set P, i.e., $n_P(p, r) := \lvert\mathcal{N}_P(p, r)\rvert$. Note that if $p \in P$, then $n_P(p, r)$ cannot be zero.
α	Locality parameter.
$\hat{n}_{\mathbb{P},\mathbb{R}}(p, r, \alpha)$	Average of $n_{\mathbb{R}}(p, \alpha r)$ over the set of r-neighbours of $p \in \mathbb{P}$, i.e., $$\hat{n}_{\mathbb{P},\mathbb{R}}(p, r, \alpha) := \frac{\sum_{q \in \mathcal{N}_{\mathbb{P}}(p,r)} n_{\mathbb{R}}(q, \alpha r)}{n_{\mathbb{P}}(p, r)}$$ For brevity, we often use $\hat{n}$ instead of $\hat{n}_{\mathbb{P},\mathbb{R}}$.
$\hat{\sigma}_{\mathbb{P},\mathbb{R}}(p, r, \alpha)$	Standard deviation of $n_{\mathbb{R}}(p, \alpha r)$ over the set of r-neighbours of $p \in \mathbb{P}$, i.e., $$\hat{\sigma}_{\mathbb{P},\mathbb{R}}(p, r, \alpha) := \sqrt{\frac{\sum_{q \in \mathcal{N}_{\mathbb{P}}(p,r)} \left(n_{\mathbb{R}}(q, \alpha r) - \hat{n}_{\mathbb{P},\mathbb{R}}(p, r, \alpha)\right)^2}{n_{\mathbb{P}}(p, r)}}$$ where $p \in \mathbb{P}$. For brevity we often use $\hat{\sigma}$ instead of $\hat{\sigma}_{\mathbb{P},\mathbb{R}}$.
k_σ	Determines what is *significant* deviation, i.e., a point $p \in \mathbb{P}$ is flagged as an outlier with respect to the set $\mathbb{R}$ iff $$\lvert\hat{n}_{\mathbb{P},\mathbb{R}}(p, r, \alpha) - n_{\mathbb{R}}(p, \alpha r)\rvert > k_\sigma \hat{\sigma}_{\mathbb{P},\mathbb{R}}(p, r, \alpha)$$ Typically, $k_\sigma = 3$.

Table 8.1: Symbols and definitions.

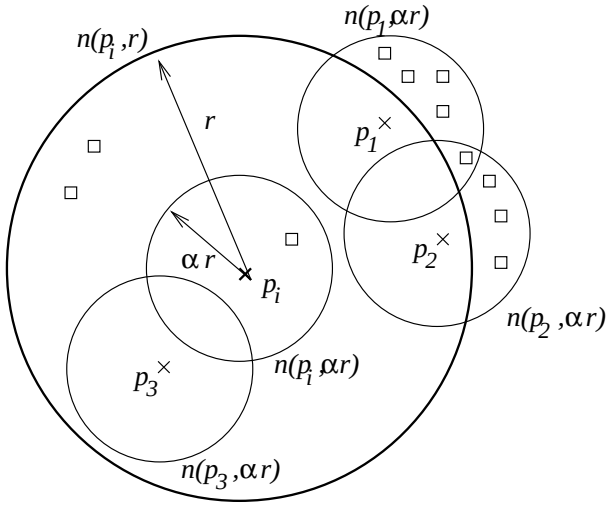


Figure 8.1: Definitions for n and $\hat{n}$. Points in the primary set $\mathbb{P}$ are shown with "$\times$" and points in the reference set $\mathbb{R}$ with "$\square$". For instance, $n_{\mathbb{P}}(p_i, r) = 4$ (including itself), $n_{\mathbb{R}}(p_i, \alpha r) = 1$, $n_{\mathbb{R}}(p_1, \alpha r) = 6$ and $\hat{n}_{\mathbb{P},\mathbb{R}}(p_i, r, \alpha) = (1 + 5 + 4 + 0)/4 = 3.25$.

Definition 17 (Counting and sampling neighborhood). *The* counting neighbourhood *(or αr-neighbourhood) is the neighbourhood of radius αr, over which each $n_{\mathbb{R}}(q, \alpha r)$ is estimated. The* sampling neighbourhood *(or r-neighbourhood) is the neighbourhood of radius r, over which we collect samples of $n_{\mathbb{R}}(q, \alpha r)$ in order to estimate $\hat{n}_{\mathbb{P},\mathbb{R}}(p, r, \alpha)$. The* locality parameter *is α.*

The locality parameter α determines the relationship between the size of the sampling neighbourhood and the counting neighbourhood. We typically set this value to $\alpha = 1/2$ (see also Section 8.4.1).

Our outlier detection scheme relies on the standard deviation of the αr-neighbour count of points in the reference set $\mathbb{R}$. Therefore, we also define the quantity $\hat{\sigma}_{\mathbb{P},\mathbb{R}}(p, r, \alpha)$ to be precisely that, for each point $p \in \mathbb{P}$ and each sampling radius r.

Definition 18 (Cross-outlier criterion). *A point $p \in \mathbb{P}$ is a* cross-outlier at scale (or radius) r *with respect to the reference set $\mathbb{R}$ if*

$$|\hat{n}_{\mathbb{P},\mathbb{R}}(p, r, \alpha) - n_{\mathbb{R}}(p, \alpha r)| > k_\sigma \hat{\sigma}_{\mathbb{P},\mathbb{R}}(p, r, \alpha)$$

Finally, the average and standard deviation with respect to radius r can provide very useful information about the vicinity of a point.

Definition 19 (Distribution plot). *For any point $p \in \mathbb{P}$, the plot of $n_{\mathbb{R}}(p, \alpha r)$ and $\hat{n}_{\mathbb{P},\mathbb{R}}(p, r, \alpha)$ with $\hat{n}_{\mathbb{P},\mathbb{R}}(p, r, \alpha) \pm 3\hat{\sigma}_{\mathbb{P},\mathbb{R}}(p, r, \alpha)$, versus r (for a range of radii of interest), is called its* (local)

Symbol	Definition
$\mathcal{C}(p, r, \alpha)$	Set of cells in some grid, with cell side $2\alpha r$, each fully contained within $\mathcal{L}_\infty$-distance r from point p.
C_i	Cell in some grid.
$c_i P$	The count of points *from set* P within the corresponding cell C_i.
$S_q P(p, r, \alpha)$	Sum of box counts (from set P) to the q-th power, i.e., $$S_q P(p, r, \alpha) := \sum_{C_i \in \mathcal{C}(p,r,\alpha)} c_i P^q$$
$P^q_{P,R}(p, r, \alpha)$	Sum of box count products (from sets P and R); in particular, $$P^q_{P,R}(p, r, \alpha) := \sum_{C_i \in \mathcal{C}(p,r,\alpha)} c_i P c_i R^q$$ Note that, $S_q P = P^{q-1}_{P,P}$.

Table 8.2: Box-counting symbols and definitions.

distribution plot.

8.2.2 Advantages of our definitions

Among several alternatives for an outlier score (such as $\max(\hat{n}/n, n/\hat{n})$, to give one example), our choice allows us to use probabilistic arguments for flagging outliers.

The above definitions and concepts make minimal assumptions. The only general requirement is that a distance is defined. Arbitrary distance functions are allowed, which may incorporate domain-specific, expert knowledge, if desired.

A final but very important point is that distance distributions can be quickly estimated in time that is linear with respect both to dataset sizes and dimensionality. Therefore, the above definitions lend themselves to fast, single-pass estimation algorithms, based on box-counting (see Chapter 3). The only further constraint imposed in this case is that all points must belong to a k-dimensional vector space (either inherently, or after employing some embedding technique).

The main idea is to approximate the r-neighbour counts for each point p with pre-computed counts of points within a cell[1] of side r which contains p.

[1]In practice, we have to use multiple cells in a number randomly shifted grids and use some selection or voting scheme to get a good approximation; see Chapter 3 for more details.

In a little more detail, in order to quickly estimate $\hat{n}(p, r, \alpha)$ for a point $p_i \in \mathbb{P}$ (from now on, we assume $\mathcal{L}_\infty$ distances), we can use the following approach. Consider a grid of cells with side $2\alpha r$ over both sets $\mathbb{P}$ and $\mathbb{R}$. Within each cell, we store separate counts of points it contains from $\mathbb{P}$ and $\mathbb{R}$. Perform a *box count* on the grid: For each cell C_j in the grid, find the counts, $c_j\mathbb{R}$ and $c_j\mathbb{P}$, of the number of points from $\mathbb{R}$ and $\mathbb{P}$, respectively, in the cell. There is a total number of $c_j\mathbb{P}$ points $p \in \mathbb{P} \cap C_j$ (counting p itself), each of which has $c_j\mathbb{P}$ neighbours from $\mathbb{R}$. So, the total number of $\mathbb{R}$ neighbours over all points from $\mathbb{P}$ in C_j is $c_j\mathbb{P}c_j\mathbb{R}$. Denote by $\mathcal{C}(p, r, \alpha)$ the set of all cells in the grid such that the entire cell is within distance r of p_i. We use $\mathcal{C}(p, r, \alpha)$ as an approximation for the r-neighbourhood of p_i. Summing over all these cells, we get a total number of $\mathbb{P}$-$\mathbb{R}$ pairs of $P_{\mathbb{P},\mathbb{R}}(p, r, \alpha) := \sum_{C_j \in \mathcal{C}(p,r,\alpha)} c_j\mathbb{P}c_j\mathbb{R}$. The total number of objects is simply the sum of all box counts for points in $\mathbb{P}$, i.e., $S_1\mathbb{P}(p, r, \alpha)$

$$\hat{n}_{\mathbb{P},\mathbb{R}}(p, r, \alpha) = \frac{P^1_{\mathbb{P},\mathbb{R}}(p, r, \alpha)}{S_1\mathbb{P}(p, r, \alpha)}$$

A similar calculation can be done to estimate

$$\hat{\sigma}_{\mathbb{P},\mathbb{R}}(p, r, \alpha) = \sqrt{\frac{P^2_{\mathbb{P},\mathbb{R}}(p, r, \alpha)}{S_1\mathbb{P}(p, r, \alpha)} - \left(\frac{P^1_{\mathbb{P},\mathbb{R}}(p, r, \alpha)}{S_1\mathbb{P}(p, r, \alpha)}\right)^2}$$

8.3 Experimental results

In this section we give examples of our method and discuss some important observations related to our approach, as well as the problem in general.

Gap In this case (see Figure 8.2, top row) the primary set consists of 340 points with a uniformly random distribution within a square region. In single-set outlier detection ($\mathbb{R} = \mathbb{P}$) some fringe points are flagged with a positive deviation (i.e., at some scale, their neighbour count is below the local average). Also, a few interior points in locally dense regions are flagged with a negative deviation.

In cross-outlier detection, we use a reference set $\mathbb{R}$ of 1400 points, again uniformly random in a slightly larger square region, but with a central square gap. As expected, the points of $\mathbb{P}$ that fall within well within the gap of $\mathbb{R}$ are detected as cross-outliers with a positive deviation. Also, very few[2] other points are flagged.

[2]Since $\mathbb{R}$ is significantly denser than $\mathbb{P}$, this is expected.

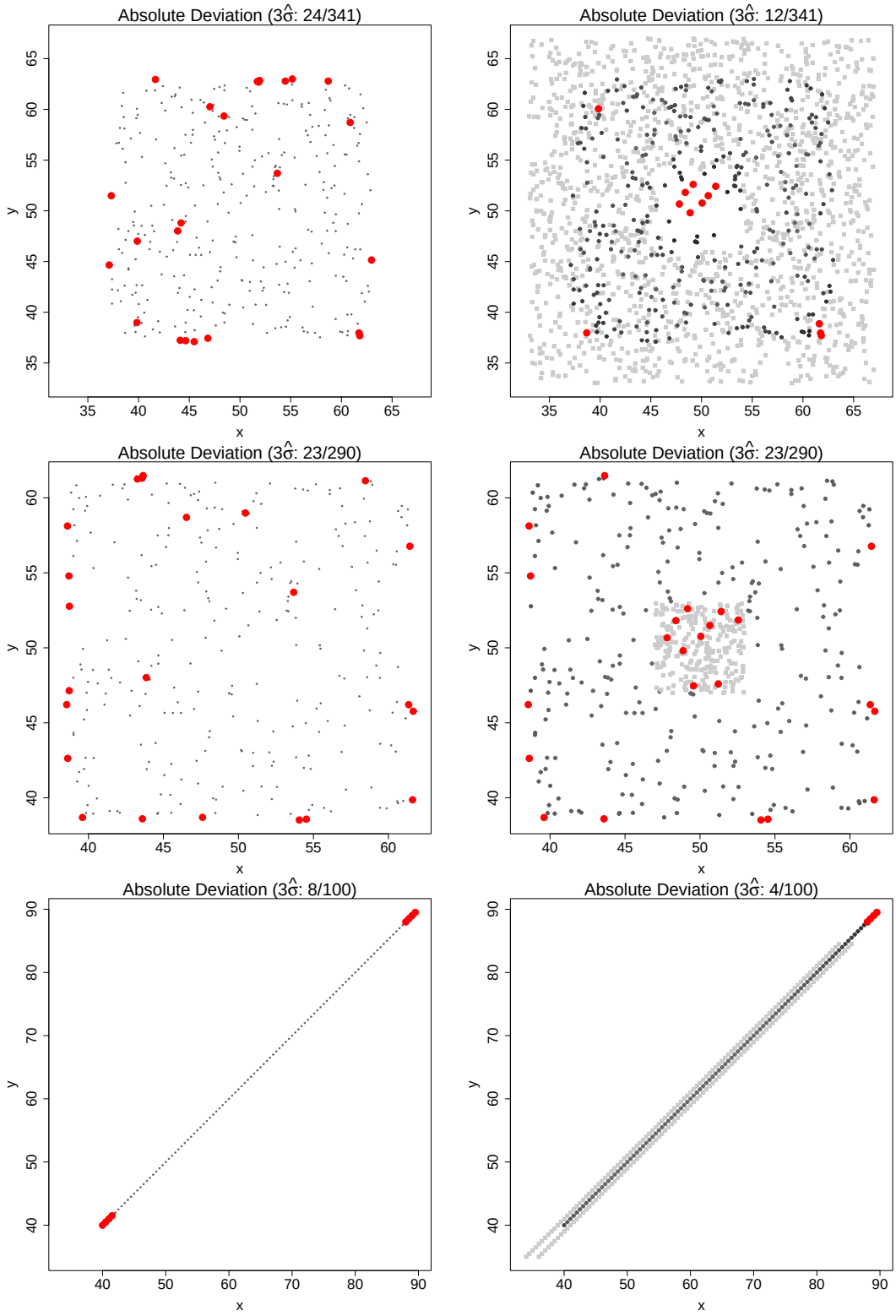


Figure 8.2: "Plain" outliers (left, $\mathbb{R} = \mathbb{P}$) and cross-outliers (right). The reference set is shown with square, grey points in the right column. Outliers are marked with larger, red points in each case. In all cases, $\alpha = 1/4$.

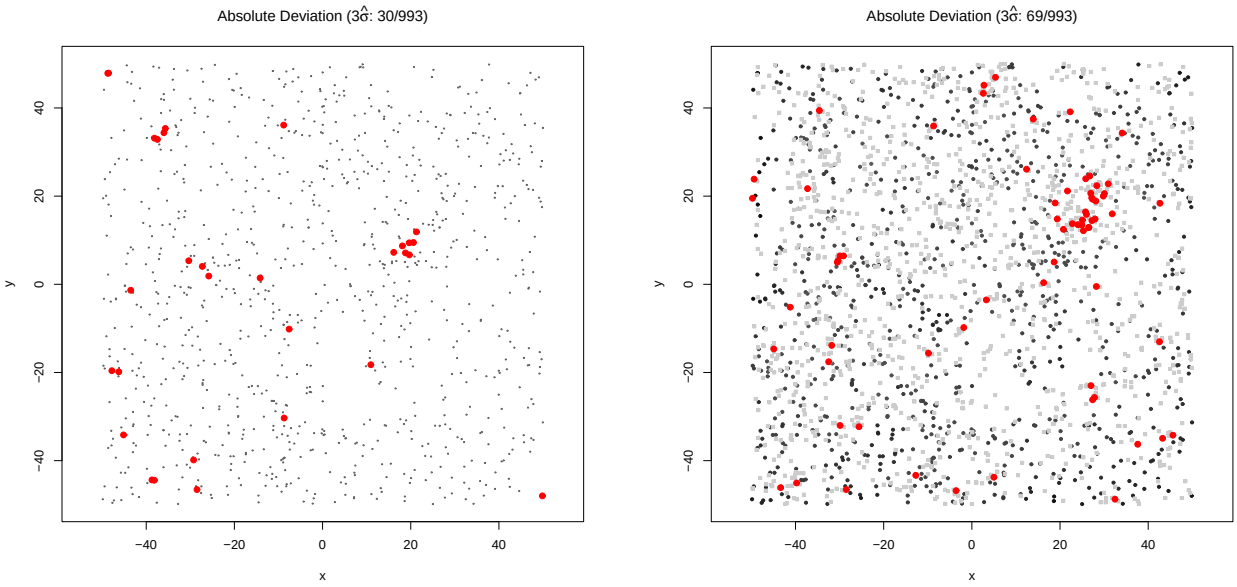


Figure 8.3: "Plain" outliers (left, $\mathbb{R} = \mathbb{P}$) and cross-outliers (right) for the galaxy datasets. In all cases, $\alpha = 1/4$.

`Core` In this case (see Figure 8.2, middle row), the primary set again consists of 300 points with a uniformly random distribution within a square region. The single-set outliers are similar to the previous case.

In cross-outlier detection, we use a reference set $\mathbb{R}$ of 250 points uniformly random within a central square "core." As expected again, the points of $\mathbb{P}$ that fall within the reference "core" are all detected as outliers. Also, some fringe points are still detected as outliers (see Section 8.3.1).

`Lines` The primary set $\mathbb{P}$ consists of 100 points regularly spaced along a line (Figure 8.2, bottom row). The single-set outliers ($\mathbb{P} = \mathbb{R}$) consist of eight points, four at each end of the line. Indeed, these points are "special," since their distribution of neighbours clearly differs from that of points in the middle of the line.

In cross outlier detection, the reference set $\mathbb{R}$ consists of two lines of 100 points each, both parallel to $\mathbb{P}$ and slightly shifted downward along their common direction. As expected, the points at the bottom-left end of $\mathbb{P}$ are no longer outliers, with respect to $\mathbb{P}$. Note that the *same* four points along the top-right end are flagged (see discussion in Section 8.3.1).

`Galaxy` The primary set consists of a section with 993 spiral galaxies and the reference set of a section with 1218 elliptical galaxies, both from the Sloan Digital Sky Survey (Figure 8.3).

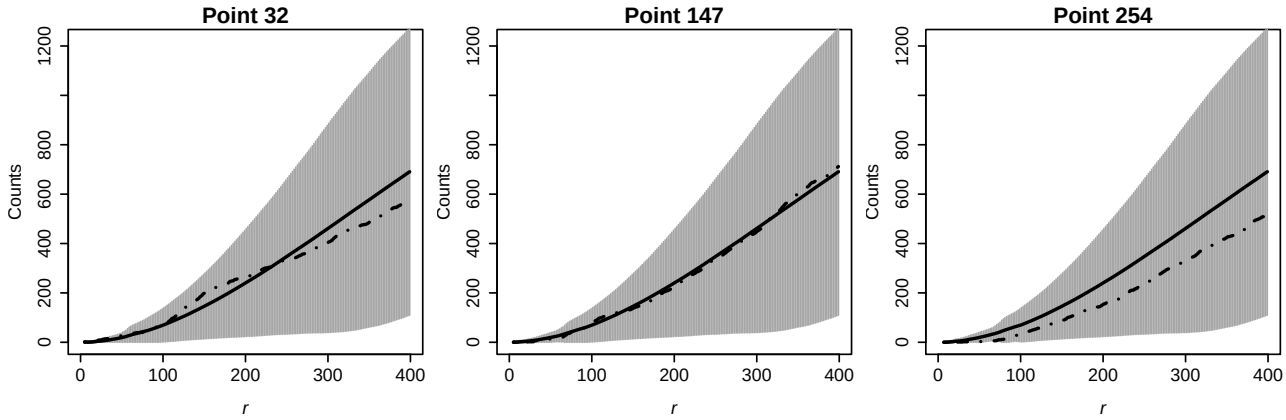


Figure 8.4: Distribution plot for cross-outliers in `Galaxy`. The horizontal axis is scale (or, sampling radius r). The solid line is $\hat{n}_{\mathbb{P},\mathbb{R}}(p, r, \alpha)$ and the dashed line is $n_{\mathbb{R}}(p, \alpha)$. The grey bands span $\pm 3\hat{\sigma}_{\mathbb{P},\mathbb{R}}(p, r, \alpha)$ around the average. The galaxy on the right is flagged with positive deviation, the other two with negative. All are flagged at small scales by a narrow margin.

Although not shown in the figure, all cross-outliers are flagged with a *negative* deviation (except two at the very edge of the dataset). Also (see Figure 8.4 and Section 8.3.1) all are flagged by a narrow margin. This is indeed expected: elliptical galaxies form clusters, intertwined with clusters of spiral galaxies. The distribution is overall even (as evidenced by the consistently wide standard deviation band); however, a few of the elliptical galaxies are within unusually dense clusters of spiral galaxies.

8.3.1 Observations

Fringe points The points located along the fringes of a data set are clearly different from the rest of the points.

One could argue that outlier definitions such as the one of the depth-based approach [JKN98] rely *primarily* on this observation in order to detect outliers. Our method goes beyond that and can also capture isolated central points (as can be seen, for example, from the `Gap` example), but can still distinguish fringe points.

With respect to pairwise distances upon which our approach is based, the first observation is that fringe points have fewer neighbours than interior points. More than that, however, all neighbours of fringe points lie on the *same* half-plane. It is a consequence of this *second* fact that the standard deviation of neighbour counts is (comparatively) smaller at certain scales for fringe points.

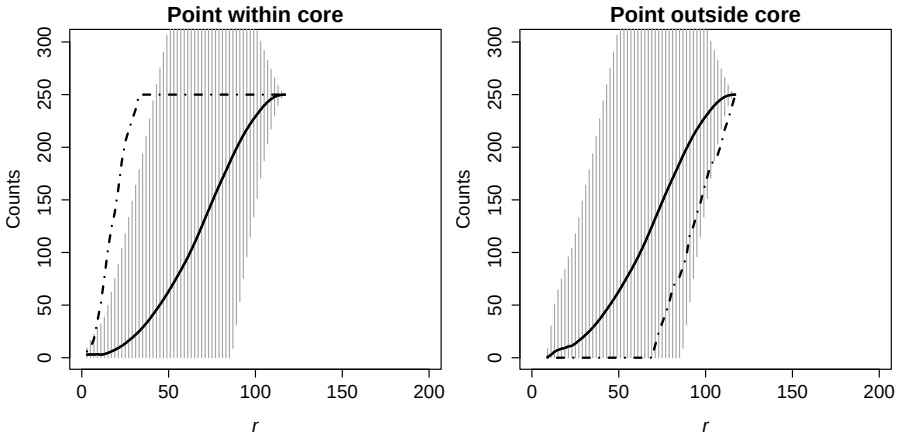


Figure 8.5: Distribution plot for cross-outliers in `Core`. Again, the horizontal axis is scale (or, sampling radius r). The solid line is $\hat{n}_{\mathbb{P},\mathbb{R}}(p, r, \alpha)$ and the dashed line is $n_{\mathbb{R}}(p, \alpha)$. The grey bands span $\pm 3\hat{\sigma}_{\mathbb{P},\mathbb{R}}(p, r, \alpha)$ around the average.

This explains why in the `Core` example more fringe points are detected as cross-outliers than in `Gap`. The reference set in `Gap` is chosen to cover a slightly larger region than the primary set in order to illustrate this point. The fringe points of $\mathbb{P}$ in `Gap` are not fringe points *with respect to* $\mathbb{R}$: they have $\mathbb{R}$-neighbours on all sides of the plane. However, the fringe points of $\mathbb{P}$ in `Core` have $\mathbb{R}$-neighbours only on one half-plane. Thus, the fringe points of $\mathbb{P}$ in `Core` are indeed different than the interior points (always *with respect to* $\mathbb{R}$).

Role of each distribution In this paragraph we further discuss the sampling and counting neighbourhoods. In particular, the former contains points of the primary set $\mathbb{P}$, while the latter of the reference set $\mathbb{R}$. Thus, the distribution of points in *both* sets plays an important role in cross-outlier detection (but see also Section 8.4.1).

This explains the fact that in `Lines` the same four endpoints are flagged as cross-outliers. We argue that this is a desirable feature. First, the points near the top-right end that are closer to $\mathbb{R}$ are indeed less "distinct" than their neighbours at the very end. This fact depends on the distribution of $\mathbb{P}$, not $\mathbb{R}$! Furthermore, consider extending $\mathbb{P}$ toward the top-right: then, neither of the endpoints are suspicious (whether surrounded or not by points of $\mathbb{R}$). This, again, depends on the distribution of $\mathbb{P}$! Indeed, in the latter case, our method does not detect any outliers.

Digging deeper As hinted in the discussion of the results, the sign of the deviation can give us important information. However, we can go even further and examine the *distri-*

bution plots, which we discuss very briefly here. Figure 8.5 is included as an example. We can clearly see that a point within the gap belongs to a sparse region (with respect to $\mathbb{R}$). Moreover, we can clearly see that the point within the gap is flagged by a much wider margin and at a wider range of scales, whereas a fringe point is marginally flagged. Thus, the distribution plots provide important information about *why* each point is an outlier, as well as its vicinity.

8.4 Discussion

In this section we first discuss why the problem of cross-outlier detection is different from the single-set case, even though the two may, at first, seem almost identical. We also discuss some directions for future research. These relate to the fast, single-pass estimation algorithms that our definitions admit.

8.4.1 Differences to single class outlier detection

The intuitive definition of [Haw80] implies two important parts in any definition of an outlier: *what* is considered a deviation (i.e., where or how we look for them) and *how* do we determine *significant* deviations. Therefore, all outlier definitions employ a model for the data and a measure of correlation, either explicitly or implicitly.

The first difference in the case of cross-outliers follows directly from the problem definition. What we essentially estimate is not a single probability distribution or correlation, but either some (conditional) probability with respect to the reference set or the covariance among sets. However, several of the existing definitions do not make their model assumptions clear or employ a model that cannot be easily extended as described above. These outlier detection approaches are hard to modify.

It should be noted that our definition employs a very general and intuitive model which is based on pairwise distanced and makes minimal assumptions.

The second major difference again follows from the fact that we are dealing with two *separate* sets. Simply put, in the "classical" case ($\mathbb{R} = \mathbb{P}$), we can obviously assume that a point set is co-located in space with respect to itself. However, this need not be the case when $\mathbb{R} \neq \mathbb{P}$. This assumption is sometimes implicitly employed in outlier definitions.

Tools such as that of [TTPF01] are useful here as a first step to determine the *overall* spatial relationship between the two sets. It must further be noted that, in our approach, the

locality parameter α is tunable and typically two values should be sufficient: $\alpha \approx 1/2$ (or 1) and any $\alpha \leq r_{min} / \max\{R_{\mathbb{P}}, R_{\mathbb{R}}\}$ where r_{min} is the smallest distance between any two points (irrespective of type)[3].

8.4.2 Efficiency considerations

Our definitions are based on pairwise distance distributions. As demonstrated in [TTPF01, PKGF03], these can be estimated very quickly with a single pass over the data, in time that is practically linear with respect to both data set size and dimensionality. The only minor restriction imposed by these algorithms is that $\alpha = 1/2^k$ for some integer k.

Furthermore, if we have more than two classes of points, the pre-processing step for box counting can be modified to keep separate counts for each class. This does not increase computational cost (only space in proportion to the number of classes) and allows fast outlier detection where the reference set $\mathbb{R}$ is the *union* of points from several classes (rather than a single class).

8.4.3 Generalisations

The observation in the last paragraph of the previous section naturally leads to the problem of multi-class outlier detection. As pointed out, the fast algorithms can easily detect outliers when the reference set $\mathbb{R}$ is any *given* combination of classes, without incurring any extra computational cost.

An interesting future research direction is to extend these algorithms with heuristic pruning approaches (e.g., similar to those in association rule[4] algorithms [AS94]; in our case, items correspond to point classes) to efficiently search the entire space of all class combinations (i.e., point-set unions) in the place of $\mathbb{R}$.

8.5 Conclusions

In this chapter we presented the problem of *cross-outlier* detection. This is the first contribution; we argue that this is a non-trivial problem of practical interest and certainly more than

[3]The second choice for α formally implies that, at every scale, the sampling neighbourhood completely covers both datasets.

[4]This is one potential approach; regions with *no* co-located classes can probably be ignored. Of course, this far from exhausts all possible pruning techniques.

an immediate generalization. We discuss several aspects of the problem that make it different from "classical" outlier detection. The former is a special case of cross-outliers (with $\mathbb{R} = \mathbb{P}$) but the converse is not true.

Beyond introducing the problem, we present a method that can provide an answer. Furthermore, our definitions use a statistically intuitive flagging criterion and lend themselves to fast, single-pass estimation. We demonstrate our approach using both synthetic and real datasets.

Chapter 9

Simultaneous spatial and feature clustering

In this chapter we deal with the problem of finding spatial correlation patterns and feature co-occurrence patterns, *simultaneously* and *automatically*. For example, consider environmental data where spatial locations correspond to patches (cells in a rectangular grid) and features correspond to species presence information. For each patch and species pair, the observed value is either one or zero, depending on whether the particular species was observed or not at that patch. In this case, feature co-occurrence patterns would correspond to species co-habitation and spatial correlation patterns would correspond to natural habitats for species groups. Combining the two will generate homogeneous regions characterised by a set of species that live in those regions. We wish to find "good" patterns of this form *simultaneously* and *automatically*.

Spatial data in this form (binary features over a set of locations) occur naturally in several settings, e.g.:

- Biodiversity data, such as the example above.
- Geographical data, e.g., presence of facilities (shops, hospitals, houses, offices, etc) over a set of city blocks.
- Environmental data, e.g., occurrence of different phenomena (storms, hurricanes, snow, drought, etc. or) over a set of locations in satellite images.
- Historical/linguistic data, e.g., occurrence of different words in different counties, or occurrence of various types of historical events over a set of locations.

In all these settings, we would like to discover meaningful feature co-occurrence and spatial

correlation patterns. Existing methods either discover one of the two types of patterns in isolation, or require the user to specify certain parameters or thresholds.

We view the problem from the perspective of succinctly summarising (i.e., compressing) the data, and we employ the Minimum Description Length (MDL) principle to automate the process. We group locations and features *simultaneously*: feature co-occurrence patterns help us compress spatial correlation patterns better, and vice versa. Furthermore, for location groups, we incorporate spatial affinity by compressing regions in a natural way.

Section 9.1 presents some of the background, in the context of our problem. Section 9.2 builds upon this background, leading to the proposed approach described in Section 9.3. Section 9.4 presents experiments that illustrate the results of our approach. Section 9.5 surveys related work. Finally, in Section 9.6 we conclude.

9.1 Background

In this section we introduce some of the background, in the context of the problem we wish to solve. In subsequent sections, we explain how we adapt these techniques for our purposes.

9.1.1 Minimum description length (MDL)

In this section we give a brief overview of a practical formulation of the minimum description length (MDL) principle. For further information see, e.g., [CT91, Grü05]. Intuitively, the main idea behind MDL is the following: Let us assume that we have a family $\mathcal{M}$ of *models* with varying degrees of complexity. More complex models $M \in \mathcal{M}$ involve more parameters but, *given* these parameters (i.e., the model $M \in \mathcal{M}$), we can describe the observed data more concisely.

As a simple, concrete example, consider a binary sequence $D := [d(1), d(2), \ldots, d(n)]$ of n coin tosses. A simple model $M^{(1)}$ might consist of specifying the number h of heads. Given this model $M^{(1)} \equiv \{h/n\}$, we can encode the dataset D using $L(D|M^{(1)}) := nH(h/n)$ bits [RL79], where $H(\cdot)$ is the Shannon entropy function. However, in order to be fair, we should also include the number $L(M^{(1)})$ of bits to transmit the fraction h/n, which can be done using $\log^\star n$ bits for the denominator and $\lceil \log(n+1) \rceil$ bits for the numerator $h \in \{0, 1, \ldots, n\}$, for a total of $L(M^{(1)}) := \log^\star n + \lceil \log(n+1) \rceil$ bits. In the above, $\log^\star$ is the

universal code length for integers [Ris83] and is written as

$$\log^\star x = \log_2 x + \log_2 \log_2 x + \ldots,$$

where only the positive terms are retained.

Definition 20 (Code length and description complexity). $L(D|M^{(1)})$ *is* code length *for* D*, given the model* $M^{(1)}$*.* $L(M^{(1)})$ *is the* model description complexity *and* $L(D, M^{(1)}) := L(D|M^{(1)}) + L(M^{(1)})$ *is the* total code length*.*

A slightly more complex model might consist of segmenting the sequence in two pieces of length $n_1 \geq 1$ and $n_2 = n - n_1$ and describing each one independently. Let h_1 and h_2 be the number of heads in each segment. Then, to describe the model $M^{(2)} \equiv \{h_1/n_1, h_2/n_2\}$, we need $L(M^{(2)}) := \log^\star n + \lceil \log n \rceil + \lceil \log(n - n_1) \rceil + \lceil \log(n_1 + 1) \rceil + \lceil \log(n_2 + 1) \rceil$ bits. Given this information, we can describe the sequence using $L(D|M^{(2)}) := n_1 H(h_1/n_1) + n_2 H(h_2/n_2)$ bits.

Now, assume that our family of models is $\mathcal{M} := \{M^{(1)}, M^{(2)}\}$ and we wish to choose the "best" one for a particular sequence D. We will examine two sequences of length $n = 16$, both with 8 zeros and 8 ones, to illustrate the intuition.

Let $D_1 := \{0, 1, 0, 1, \cdots, 0, 1\}$, with alternating values. We have $L(D_1|M_1^{(1)}) = 16H(1/2) = 16$ and $L(M_1^{(1)}) = \log^\star 16 + \lceil \log(16 + 1) \rceil = 10 + 5 = 15$. However, for $M_1^{(2)}$ the best choice is $n_1 = 15$, with $L(D_1|M_1^{(2)}) \approx 15$ and $L(M_1^{(2)}) \approx 19$. The *total* code lengths are $L(D_1, M_1^{(1)}) \approx 16 + 15 = 31$ and $L(D_1, M_1^{(2)}) \approx 15 + 19 = 34$. Thus, based on total code length, the simpler model is better[1]. The more complex model may give us a lower code length, but that benefit is not enough to overcome the increase in description complexity: D_1 does not exhibit a pattern that can be exploited *by a two-segment model* to describe the data.

Let $D_2 := \{0, \cdots, 0, 1, \cdots, 1\}$ with all similar values contiguous. We have again $L(D_2|M_2^{(1)}) = 16$ and $L(M_2^{(1)}) = 15$. But, for $M_2^{(2)}$ the best choice is $n_1 = n_2 = 8$ so that $L(D_2|M_2^{(2)}) = 8H(0) + 8H(1) = 0$ and $L(M_2^{(2)}) \approx 24$. The *total* code lengths are $L(D_2, M_2^{(1)}) \approx 16 + 15 = 31$ and $L(D_2, M_2^{(2)}) \approx 0 + 24 = 24$. Thus, based on total code length, the two-segment model is better. Intuitively, it is clear that D_2 exhibits a pattern that can help reduce the total code length. This intuitive fact is precisely captured by the total code length.

In fact, this simple example is prototypical of the groupings we will consider later. More generally, we could consider a family $\mathcal{M} := \{M^{(k)} \mid 1 \leq k \leq n\}$ of k-segment models and

[1]The absolute codelengths are not important; the bit overhead compared to the straight transmission of D tends to zero, as n grows to infinity.

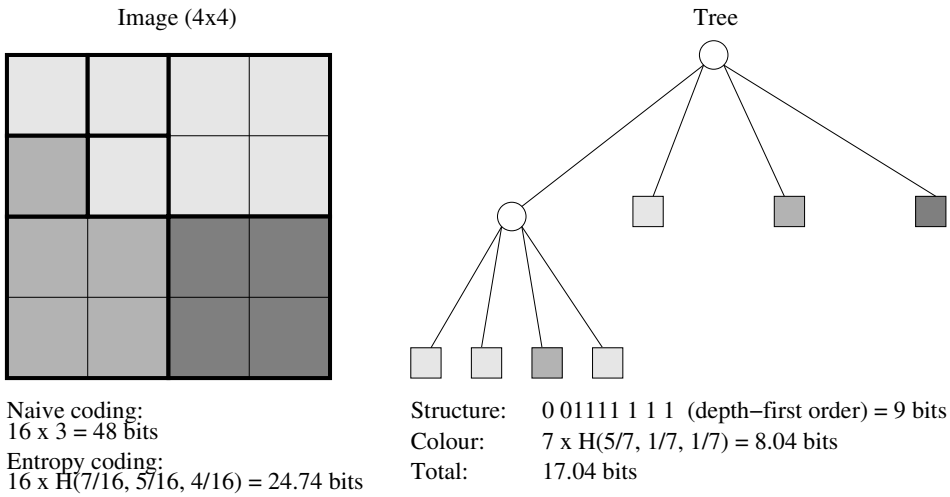


Figure 9.1: Quadtree compression: The map on the left has $4 \times 4 = 16$ cells (pixels), each having one of three possible values. The resulting quadtree has 10 leaf nodes, again each having one of three possible values.

apply the same principles. Furthermore, the datasets we will consider are two-dimensional matrices $D := [d(i,j)]$, instead of one-dimensional sequences. In Section 9.2.2 we address both of these issues. To complicate matters even further, one of the dimensions of D has a spatial location associated with it. Section 9.3 presents data description models that also incorporate this information.

In fact, choosing the appropriate family of models is non-trivial. Roughly speaking, at one extreme we have the singleton family of "just the raw data," which cannot describe any patterns. At the other extreme, we have "all Turing machine programs that produce the data as output," which can describe the most intricate patterns, but make model selection intractable. Striking the right balance is a challenge. In this chapter, we address it for the case of spatial data.

9.1.2 Quadtree compression

A quadtree is a data structure that can be used to efficiently index contiguous regions of variable size in a grid. It has been used successfully in image coding and has the benefit of small overhead and very efficient construction [VG87]. Figure 9.1 shows a simple example. Each internal node in a quadtree corresponds to a partitioning of a rectangular region into four quadrants. The leaf nodes of a quadtree represent rectangular groups of cells and have a value p associated them, where p is the group ID. In the following we briefly describe quadtree codelengths.

Structure The structure of a quadtree uniquely corresponds to a partitioning of the grid. For example, the partitioning into three regions in Figure 9.1 on the left corresponds to the structure on the right. This partitioning is chosen in a way that respects spatial correlations. The structure can be described easily by performing a traversal of the tree and transmitting a zero for non-leaf nodes and a one for leaf nodes. The traversal order is not significant; we choose depth-first order (see Figure 9.1).

Values Quadtree structure conveys information about the partition boundaries (thick grid lines in Figure 9.1). These capture all correlations: in effect, we have reduced the original set of equal-sized cells to a (smaller) set of variable-sized, square cells (each one corresponding to a leaf node in the quadtree). Since the correlations have already been taken into account, we may assume that the leaf node values are independent. Therefore, the cost to transmit the values is equal to the total number of leaf nodes, multiplied by the entropy of the leaf value distribution.

Lemma 6 (Quadtree codelength). *Let T be a quadtree with m' leaf nodes, of which m'_p have value p, where $1 \leq p \leq k$. Then, the number of internal nodes is $\lceil m'/3 \rceil - 1$. Structure information can be transmitted using one bit per node (leaf/non-leaf) and values can be transmitted using entropy coding. Therefore, the corresponding total codelength is*

$$L(T) = m' H\left(\frac{m'_1}{m'}, \frac{m'_2}{m'}, \ldots, \frac{m'_k}{m'}\right) + \left\lceil \frac{4m'}{3} \right\rceil - 1$$

This has a straightforward but important consequence:

Lemma 7. *The codelength $L(T)$ for a quadtree T can be computed in constant time, if we know the distribution of leaf node values.*

In other words, for a full quadtree (i.e., one where each node has either zero or four descendants), if we know m' and m'_p, for $1 \leq p \leq k$, we can compute the cost in closed form, using Lemma 6. Note that the quadtree does not have to be perfect (i.e., all leaves do not have to be at the same level). When a node is reassigned a different value, region consolidations may occur (i.e., pruning of leaves with same value). Updating m' and m'_p will require time proportional to the number of consolidations, which are typically localised. In the worst case, the time will be $O(\log m)$ if pruning cascades up to the root node.

Symbol	Definition
D	Binary data matrix.
m, n	Dimensions of D (rows, columns); rows correspond to cells.
k, ℓ	Number of row and column groups.
k^*, ℓ^*	Optimal number of groups.
Q_X, Q_Y	Row and column assignments to groups.
$D_{p,q}$	Submatrix for intersection of p-th row and q-th column group.
m_p, n_q	Dimensions of $D_{p,q}$.
$\|D_{p,q}\|$	Number of elements $\|D_{p,q}\| := m_p n_q$.
$\rho_{p,q}$	Density of 1s in $D_{p,q}$.
$H(\cdot)$	Binary Shannon entropy function.
$L(D_{p,q}\|Q_X, Q_Y, k, \ell)$	Codelength for $D_{p,q}$.
$L(D, Q_X, Q_Y, k, \ell)$	Total codelength for D.

Table 9.1: Symbols and definitions.

9.2 Preliminaries

In this section we formalise the problem and prepare the ground for introducing our approach in Section 9.3.

9.2.1 Problem definition

Assume we are given m cells on an evenly-spaced grid (e.g., field patches in biological data) and n features (e.g., species). For each pair (i, j), $1 \leq i \leq m$ and $1 \leq j \leq n$, we are also given a binary observation (e.g., species presence/absence at each cell).

We want to group both cells and features, thus also implicitly forming groups of observations (each such group corresponding to an intersection of cell and feature groups). The two main requirements are:

1. Spatial affinity: Groups of cells should exhibit spatial coherence, i.e., if two cells i_1 and i_2 are close together, then we wish to favour cell groupings that place them in the same group. Furthermore, spatial affinity should be balanced with feature affinity in a principled way.

2. Homogeneity: The implicit groups of observations should be as homogeneous as possible, i.e., be nearly all-ones or all-zeros.

The problem and our proposed solution can be easily extended to a collection of categorical features (i.e., taking more than two values, from a finite set of possible values) per cell.

9.2.2 MDL and binary matrices

Let $D = [d(i,j)]$ denote a $m \times n$ ($m, n \geq 1$) binary data matrix. A *bi-grouping* is a simultaneous grouping of the m rows and n columns into k and ℓ disjoint row and column groups, respectively. Formally, let

$$Q_X : \{1, 2, \ldots, m\} \rightarrow \{1, 2, \ldots, k\}$$
$$Q_Y : \{1, 2, \ldots, n\} \rightarrow \{1, 2, \ldots, \ell\}$$

denote the assignments of rows to row groups and columns to column groups, respectively. The pair $\{Q_X, Q_Y\}$ is a bi-grouping.

Based on the observation that a good compression of the matrix implies a good, concise grouping, both k, ℓ as well as the assignments Q_X, Q_Y can be determined by optimising the description cost of the matrix. Let

$$R_p := Q_X^{-1}(p), \quad 1 \leq p \leq k$$
$$C_q := Q_Y^{-1}(q), \quad 1 \leq q \leq \ell$$

be the set of rows and columns assigned to row group p and column group q, with sizes $m_p := |R_p|$ and $n_q := |C_q|$, respectively. Then, let

$$D_{p,q} := [d(R_p, C_q)], \quad 1 \leq p \leq k, 1 \leq q \leq \ell.$$

be the sub-matrix of D defined by the intersection of row group p and column group q. The total codelength $L(D) \equiv L(D, Q_X, Q_Y, k, \ell)$ for transmitting D is expressed as

$$L(D) = L(D|Q_X, Q_Y, k, \ell) + L(Q_X, Q_Y, k, \ell).$$

For the first part of Eq. 9.2.2, elements within each $D_{p,q}$ are assumed to be drawn independently, so that

$$L(D_{p,q}|Q_X, Q_Y, k, \ell) = \lceil \log(|D_{p,q}| + 1) \rceil + |D_{p,q}| H\big(\rho_{p,q}\big)$$

where $\rho_{p,q}$ is the density (i.e., probability) of ones within $D_{p,q}$ and $|D_{p,q}| = m_p n_q$ is the number of its elements. This is analogous to the coin toss sequence models described in

Section 9.1.1. Finally,

$$L(D|Q_X, Q_Y, k, \ell) := \sum_{p=1}^{k} \sum_{q=1}^{\ell} L(D_{p,q}|Q_X, Q_Y, k, \ell).$$

For the second part of Eq. 9.2.2, row and column groupings are assumed to be independent, hence

$$\begin{aligned} L(Q_X, Q_Y, k, \ell) &= L(Q_X, Q_Y|k, \ell) + L(k, \ell) \\ &= L(Q_X|k) + L(k) + L(Q_Y|\ell) + L(\ell). \end{aligned}$$

Finally, a uniform prior is assigned to the number of groups, as well as to each possible grouping given the number of groups, i.e.,

$$\begin{aligned} L(k) &= -\log \Pr(k) = \log m \\ L(Q_X|k) &= -\log \Pr(Q_X|k) = \log \binom{m}{m_1 \ \cdots \ m_k} \end{aligned}$$

and similarly for the column groups.

Using Stirling's approximation $\ln n! \approx n \ln n - n$ and the fact that $\sum_i m_i = m$, we can easily derive the bound

$$\begin{aligned} L(Q_X|k) &= \log \binom{m}{m_1 \ \cdots \ m_k} = \log \frac{m!}{m_1! \cdots m_k!} \\ &= \log m! - \sum_{i=1}^{k} \log m_i! \approx m \log m - \sum_{i=1}^{k} m_i \log m_i \\ &= -m \sum_{i=1}^{k} \tfrac{m_i}{m} \log \tfrac{m_i}{m} = mH\left(\tfrac{m_1}{m}, \ldots, \tfrac{m_k}{m}\right) \\ &\leq m \log k. \end{aligned}$$

Therefore, we have the following:

Lemma 8. *The codelength for transmitting an arbitrary m-to-k mapping Q_X, where m_p symbols from the range are mapped into each value $p, 1 \leq p \leq k$, is approximately*

$$L(Q_X|k) = mH\left(\tfrac{m_1}{m}, \ldots, \tfrac{m_k}{m}\right)$$

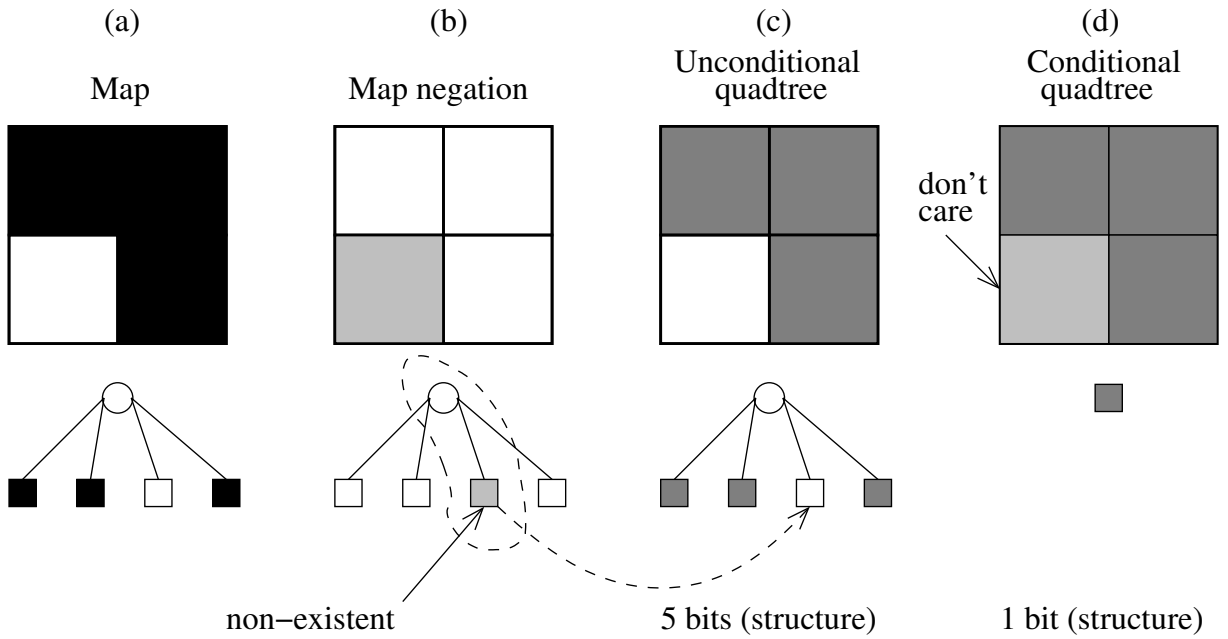


Figure 9.2: Quadtree compression to discount the complexity of the enclosing region's shape; only the complexity of cell group shapes *within* the map's boundaries matters.

9.2.3 Map boundaries

The set of all cells may form an arbitrary, complex shape, rather than a square with a side that is a power of two. However, we wish to penalise only the complexity of interior cell group boundaries. The shape of boundaries on the edges (e.g., coastline) of the map should not affect the cost.

For example, assume that our dataset consists of the three black cells in Figure 9.2(a). If all three cells belong to the same group and we encode this information naïvely, then we get a quadtree with five nodes (Figure 9.2(c)). However, the complexity of the resulting quadtree is only due to the fact that the bottom-left is "non-existent."

If we know the shape of the entire map *a priori*, we can encode the same information using 1 bit, as shown in Figure 9.2(d). In essence, both transmitter and receiver agree upon a set of "existing" cell locations (or, equivalently, a *prior* quadtree corresponding to the map description). This information should not be accounted for in the total codelength, as it is fixed for a given dataset. Given this information, all cells groups in the transmitted quadtree (e.g., group of both light and dark grey in Figure 9.2(d)) should be *intersected* with the set of existing cells (e.g., black in Figure 9.2(a)) to get the actual cells belonging to each group (e.g., only dark grey in Figure 9.2(d)).

Since the "non-existent" locations are known to both parties, we do not need to take them into account for the leaf value codelength, which is still $m'H(m'_1/m', \ldots, m'_k/m')$ (see Lemma 6), where m'_p is the number of quadtree leaves having value p ($1 \leq p \leq k$) and

$m' = \sum_{p=1}^{k} m'_p$. However, for the tree-structure codelength we need to *include* the number m'_0 of nodes corresponding to non-existent locations (e.g., white in Figure 9.2(c)). Thus, the structure codelength is $\lceil 4(m' + m'_0)/3 \rceil - 1$.

9.3 Spatial bi-grouping

In the previous sections we have gradually introduced the necessary concepts that lead up to our final goal: coming up with a simple but powerful description for binary data, which also incorporates spatial information and which allows us to automatically group both cells as well as features, without any user-specified parameters.

In order to exploit dependencies due to spatial affinity, we can pursue two alternatives:

1. Relax the assumption that the values within each $D_{p,q}$ are independent, thus modifying $L(D|Q_X, Q_Y, k, \ell)$. This amounts to saying that, *given* cells i_1 and i_2 belong to the same group, then it is more likely that feature j will be present in both cells *if* they are neighbouring.
2. Assign a non-uniform prior to the space of possible groupings, thus modifying $L(Q_X, Q_Y, k, \ell)$. This amounts to saying that two cells i_1 and i_2 are more likely to belong to the same group, *if* they are neighbouring.

We choose the latter, since our goal is to find cell groups that exhibit spatial coherence. In the former alternative, spatial affinity does not decide how we *form* the groups; it only comes into play after the groupings have been decided. The second alternative fortunately leads to efficient algorithms. Each time we consider changing the group of a cell, we have to examine how this change affects the total cost. As we shall see, this test can be performed very quickly.

In particular, we choose to modify the term $L(Q_X|k)$. Let us assume that the dataset has $m = 16$ cells, forming a 4×4 square (see Figure 9.1), and that cells are placed into $k = 3$ groups (light grey, dark grey and black in the figure). Instead of transmitting Q_X as an arbitrary m-to-k mapping (see Section 9.2.2), we can transmit the image of $m = 16$ pixels (cells), each one having one of $k = 3$ values. The length (in bits) of the quadtree for this image is precisely our choice of $L(Q_X|k)$ (compare Lemmas 6 and 8).

By using the quadtree codelength, we essentially penalise cell group region complexity, rather than the number of cell groups. The number of groups is factored into the cost indirectly, since more groups typically imply higher region complexity.

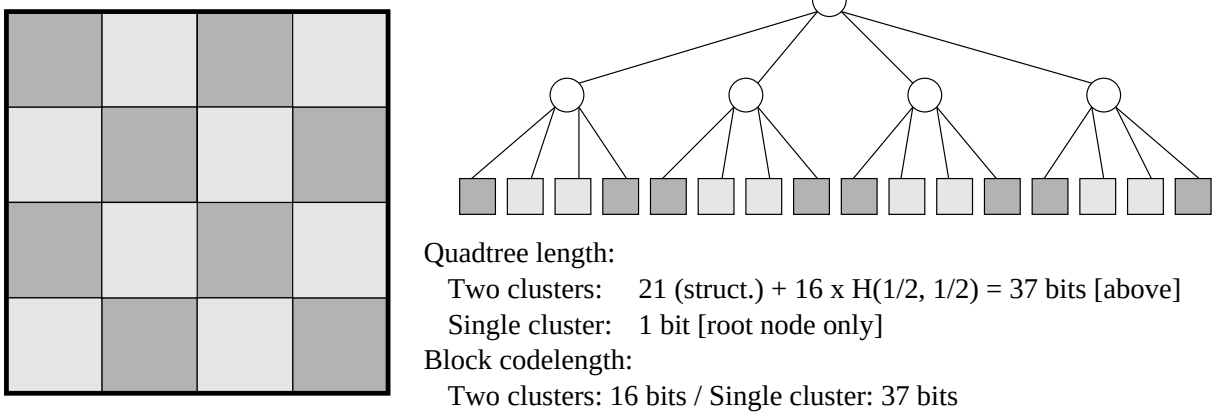


Figure 9.3: In this simple example (16 cells and 2 species, i.e., 32 binary values total), if we require groupings to obey spatial affinity, we obtain the shortest description of the dataset (locations and species) if we place all cells in one group. Any further subdivision only *adds* to the total description complexity (due to cell group region shapes).

9.3.1 Intuition

For concreteness, let us consider the case of patch locations and species presence features. The intuitive interpretation of cell and feature groups is the following:

- Row (i.e., cell) groups correspond to "neighbourhoods" or "habitats." Clearly, a habitat should exhibit a "reasonable" degree of spatial coherence.
- Column (i.e., species) groups correspond to "families." For example, a group consisting of "gull and pelican" may correspond to "seabirds," while a group with "eagle and falcon" would correspond to "mountain birds."

The patterns we find essentially summarise species and cells into families and habitats. The summaries are chosen so that the original data are compressed in the best way. Given the simultaneous summaries, we wish to make the intersection of families and habitats as uniform as possible: a particular family should either be mostly present or mostly absent from a particular habitat. This criterion jointly decides the species of a family and the cells of a habitat. However, our quad-tree based model complexity favours habitats that are spatially contiguous and do not have overly complicated boundaries.

The group search algorithms are presented in subsection 9.3.2. Intuitively, we alternatively re-group cells and features, always reducing the total codelength.

Example A simple example is shown in Figure 9.3. We choose this example as an extreme case, to clarify the trade-offs between feature and spatial affinity. Experiments based on

this boundary case are presented in section 9.4. Assume we have two species, located on a square map in a checkerboard pattern (i.e., odd cells have only species A and even cells only species B). Consider the two alternatives (we omit the number of bits to transmit species groups, which is the same in both cases):

- Two cell groups, in checkerboard pattern: One group contains only the even cells and the other only the odd cells. In this case, we need 37 bits for the quadtree (see Figure 9.3). For the submatrices, we need $\lceil \log(8 \cdot 1 + 1) \rceil + 8H(1) = 4$ bits for each of the four blocks (two species groups and two cell groups), for a total of 16 bits. The total codelength is $37 + 16 = 53$ bits.
- One cell group, containing all cells: In this case we need only 1 bit for the (singleton node) quadtree and $\lceil \log(32 \cdot 1 + 1) \rceil + 32H(1/2) = 37$ bits total for the submatrices. The total codelength is $37 + 1 = 38$ bits.

Therefore, our approach prefers to place all cells in one group. The interpretation of this is that "both species A and B occupy the same locations, with presence in $\rho_{1,1} = 50\%$ of the cells." Indeed, if we chose to perfectly separate the species instead, the cell group boundaries become overly complex and exhibit no spatial affinity. Furthermore, if the number of species was different, the tipping point in the trade-off between cell group complexity and species group "impurity" would also change. This is intuitively desirable, since describing exceptions in larger species groups is inherently more complex.

9.3.2 Algorithms

Finding a global optimum of the total codelength is computationally very expensive. Therefore, we take the usual course of employing a greedy local search (as in, e.g., standard k-means [HTF01a] or in [CPMF04]). At each step we make a local move that always reduces the objective function $L(D)$. The search for cell and feature groups is done in two levels:

- INNER level (Figure 9.4): We assume that the number of groups (for both cells and features) is *given* and try to find the grouping that minimises the total codelength. The possible local moves at this level are: (i) swapping feature vectors (i.e., group labels for rows of D), and (ii) swapping cell vectors (i.e., group labels for columns of D).
- OUTER level (Figure 9.5): Given a way to optimise for a specific number of groups (i.e., outer level), we progressively try the following local moves: (i) increase the number of cell groups, and (ii) increase the number of feature groups. Each of these moves employs the inner level search.

Algorithm INNER:

Start with an arbitrary bi-grouping (Q_X^0, Q_Y^0) of the matrix D into k row groups and ℓ column groups. Subsequently, at each iteration t perform the following steps:

1. For this step, we will hold column assignments, i.e., Q_Y^t, fixed. We start with $Q_X^{t+1} := Q_X^t$ and, for each row $i, 1 \leq i \leq n$, we update $Q_X^{t+1}(i) \leftarrow p, 1 \leq p \leq k$ so that the choice maximises the "cost gain"

$$\big(L(D|Q_X^t, Q_Y^t, k, \ell) + L(Q_X^t|k)\big) - \big(L(D|Q_X^{t+1}, Q_Y^t, k, \ell) + L(Q_X^{t+1}|k)\big).$$

   We also update the corresponding probabilities $\rho_{p,q}^{t+1}$ after each update to Q_X^{t+1}.
2. Similar to step 1, but swapping group labels of columns instead and producing a new bi-grouping (Q_X^{t+1}, Q_Y^{t+2}).
3. If there is no decrease in total cost $L(D)$, stop. Otherwise, set $t \leftarrow t + 2$, go to step 1, and iterate.

Figure 9.4: Row and column grouping, given the number of row and column groups.

If k and ℓ were known in advance, then one could use only INNER to find the best grouping. These moves guide the search towards a local minimum. In practice, this strategy is very effective. We can also perform a small number of restarts from different points in the search space (e.g., by randomly permuting rows and columns of D) and keep the best result, in terms of total codelength $L(D)$.

For each row (i.e., cell) swap, we need to evaluate the change in quadtree codelength, which takes $O(\log m)$ time in the worst case (where m is the number of cells). However, in practice, the effects of a single swap in quadtree structure tend to be local.

Complexity Algorithm INNER is linear in the number nnz of non-zeros in D. More precisely, the complexity is $O\big((nnz \cdot (k + \ell) + n \log m) \cdot T\big) = O(nnz \cdot (k + \ell + \log m) \cdot T)$, where T is the number of iterations (in practice, about 10–15 iterations suffice). We make the reasonable assumption that $nnz > n + m$. The $n \log m$ term corresponds to the quad-tree update for each row swap. In algorithm OUTER, we increase the total number $k + \ell$ of groups by one at each iteration, so the overall complexity of the search is $O((k^* + \ell^*)^2 nnz + (k^* + \ell^*) n \log m)$, which is is linear with respect to the dominating term, nnz.

Algorithm OUTER:
Start with $k^0 = \ell^0 = 1$ and at each iteration T:

1. Try to increase the number of row groups, holding the number of column groups fixed. We choose to split the row group p^* with maximum per-row entropy, i.e.,

$$p^* := \arg\max_{1 \le p \le k} \sum_{1 \le q \le \ell} |D_{p,q}| H(\rho_{p,q}) / m_p.$$

   Construct an grouping $Q_X^{T+1'}$ by moving each row i of the group p^* that will be split ($Q_X^T(i) = p^*$, $1 \le i \le m$) into the new row group $k^{T+1} = k^T + 1$, if and only if this decreases the per-row entropy of group p^*.
2. Apply algorithm INNER with initial bi-grouping $(Q_X^{T+1'}, Q_Y^T)$ to find new ones (Q_X^{T+1}, Q_Y^{T+1})..
3. If there is no decrease in total cost, stop and return $(k^*, \ell^*) = (k^T, \ell^T)$ with corresponding bi-grouping (Q_X^T, Q_Y^T). Otherwise, set $T \leftarrow T + 1$ and continue.

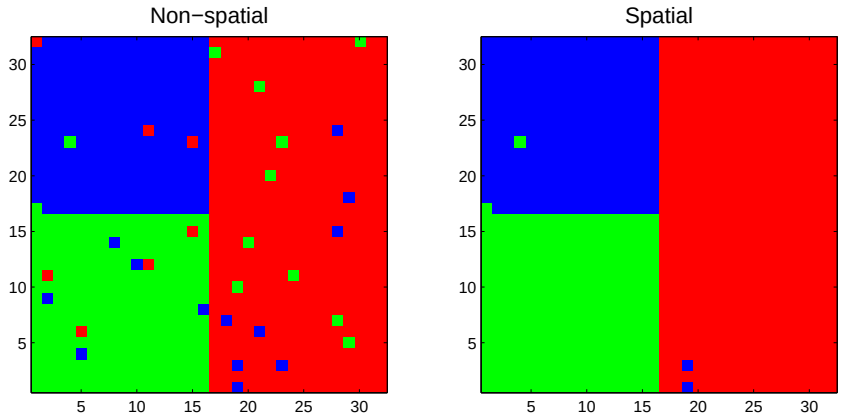
4–6. Similar to steps 1–3, but trying to increase column groups instead.

Figure 9.5: Algorithm to find number of row and column groups.

9.4 Experimental evaluation

In this section we discuss the results our method produces on a number of datasets, both synthetic (to illustrate the intuition) and real. We implemented our algorithms in Matlab 6.5. In order to evaluate the spatial coherence of the cell groups, we plot the spatial extents of each group (e.g., see also [ZK04]). In each case we compare against non-spatial bi-grouping (as presented in Section 9.2.2). This non-spatial approach produces cell groups of quality similar to or better than, e.g., straight k-means (with plain Euclidean distances on the feature bit vectors) which we also tried.

SaltPepper This is essentially the example in Section 9.3.1, with two features in a chessboard pattern. For the experiment, the map size is 32×32 cells, so the size of D is 1024×2. The spatial approach places all cells in the same group, whereas the non-spatial approach creates two row and two column groups. The total codelengths are (for a detailed explanation, see Section 9.3.1):



(a) Non-spatial grouping (b) Spatial grouping

Figure 9.6: Noisy regions.

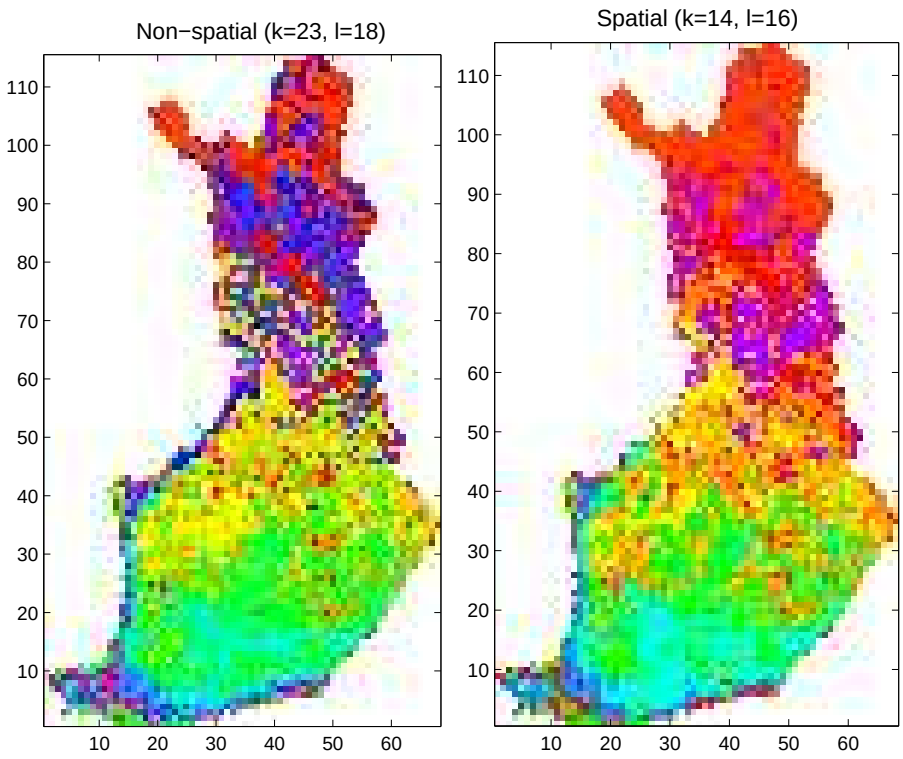
Groups	Codelength	
	Non-spatial	Spatial
1×1	2048 + 22 = 2070	2048 + 14 = 2062
2×2	0 + 61 = 61	0 + 2431 = 2431

NoisyRegions This dataset consists of three features (say, species) on a 32×32 grid, so the size of D is 1024×3. The grid is divided into three rectangles. Intuitively, each rectangle is a habitat that contains mostly one of the three species. However, some of the cells contain "stray species" in the following way: at 3% of the cells chosen at random, we placed a wrong, randomly chosen species. Figure 9.6 shows the groupings of each approach. The spatial approach favours more spatially coherent cell groups, even though they may contain some of the stray species, because that reduces the total codelength. Thus, it captures the "true habitats" almost perfectly (except for a few cells, since the algorithms find a local minimum of the codelength).

Birds This dataset consists of presence information for 219 Finnish bird species over 3813 patches, 10Km$\times$10Km in size which cover the map of Finland. The 3813×219 binary matrix contains 33.8% non-zeros (281,953 entries out of 835,047).

First, we observe that the cell groups in Figure 9.7(b) clearly exhibit a higher degree of spatial affinity than those in Figure 9.7(a). In fact, the grouping in Figure 9.7(b) captures the boreal vegetation zones in Finland: the light blue and green regions correspond to the south boreal, yellow to the mid boreal and red to the north boreal vegetation zone.

With respect to the species groups, the method successfully captures statistical outliers and biases in the data. For example, *osprey* is placed in a singleton group. The data for



(a) Non-spatial grouping ($k = 23$ cell groups, $\ell = 18$ species groups)

(b) Spatial grouping ($k = 14$ cell groups, $\ell = 16$ species groups)

Figure 9.7: Finnish bird habitats; our approach produces much more spatially coherent cell groups (see, e.g., red, purple and light blue) and captures the boreal vegetation zones.

this species was received from a special study, where a big effort was made to seek nests. Similarly, *black-throated diver* is placed in a singleton group, most likely because of its good detectability from large distances. *Rustic bunting* has highly specialised habitat requirements (mire forests) and is also not grouped with any other species.

9.5 Related work

In "traditional" clustering we seek to group only the rows of D, typically based on some notion of distance or similarity. The most popular approach is k-means (see, e.g., [HTF01a]). There are several interesting variants, which aim at improving clustering quality (e.g., k-harmonic means [ZHD00] and spherical k-means [DM01]) or determining k based on some criterion (e.g., X-means [PM00] and G-means [HE03]). Besides these, there are many other recent clustering algorithms that use an altogether different approach, e.g., CURE [GRS98], BIRCH [ZRL96], Chameleon [KHK99] and DENCLUE [HK98] (see also [HK01]). The LIMBO algorithm [ATMS04] uses a related, information theoretic approach for clustering categorical

data.

The problem of finding spatially coherent groupings is related to image segmentation; see, e.g., [ZK04]. Other more general models and techniques that could be adapted to this problem are, e.g., [BBM04, KT02, Pot52]. However, all deal only with spatial correlations and cannot be directly used for simultaneously discovering feature co-occurrences.

Prevailing graph partitioning methods are METIS [KK98] and spectral partitioning [NJW01]. Related is also the work on conjunctive clustering [MRS03] and community detection [RK01]. However, these techniques also require some user-specified parameters and, more importantly, do not deal with spatial data. Information theoretic co-clustering [DMM03] is related, but focuses on lossy compression of contingency tables, with distortion implicitly specified by providing the number of row and column clusters. In contrast, we employ MDL and a lossless compression scheme for binary matrices which also incorporates spatial information. The more recent work on cross-associations [CPMF04] is also parameter-free, but it cannot handle spatial information. Finally, Keogh et al. [KLR04a] propose parameter-free methods for classic data mining tasks (i.e., clustering, anomaly detection, classification) based on standard compression tools.

Frequent itemset mining brought a revolution [AS94] with a lot of follow-up work [HK01, HPYM04]. These techniques have also been extended for mining *spatial collocation patterns* [LMP03, Sal04, ZMCS04, HXSP03]. However, all these approaches require the user to specify a support and/or other parameters (e.g., significance, confidence, etc).

9.6 Conclusion

We propose a method to automatically discover spatial correlation and feature co-occurrence patterns. In particular:

- We group cells and features *simultaneously*: feature co-occurrence patterns help us compress spatial correlation patterns better, and vice versa.
- For cell groups (i.e., spatial correlation patterns), we propose a practical method to incorporate and exploit spatial affinity, in a natural and principled way.
- We employ MDL to discover the groupings and the number of groups, directly from the data, without any user parameters.

Our method easily extends to other natural spatial hierarchies, when available (e.g., city block, neighbourhood, city, county, state, country), as well as to categorical feature values.

Finally, we employ fast algorithms that are practically linear in the number of non-zero entries.

Chapter 10

Summary

In this part we considered mining methods when each point has one or more binary attributes (or features) associated with it, besides its spatial location (see Definition 5.

In Chapter 8 we introduced cross-outliers [PF03]. To the best of our knowledge, work on outliers up to date focuses exclusively on the problem as follows [Haw80]: "given a *single* set of observations in some space, find those that deviate so as to arouse suspicion that they were generated by a different mechanism." However, in several domains, we have more than one set of observations (or, equivalently, as single set with class labels assigned to each observation). A single observation may look normal both within its own class, as well as within the entire set of observations. However, when examined with respect to other classes, it may still arouse suspicions. Thus, we consider the problem "given a set of observations with class labels, find those that arouse suspicions, taking into account the class labels." Many of the existing outlier detection approaches cannot be extended to this case. We present one practical approach for dealing with this problem.

In Chapter 9 we consider spatial data consisting of a set of *binary features* taking values over a collection of *spatial extents* (grid cells) and we propose a method that simultaneously finds spatial correlation and feature co-occurrence patterns, without *any* parameters. In particular, we employ the Minimum Description Length (MDL) principle coupled with a natural way of compressing regions. This defines what "good" means: a feature co-occurrence pattern is good, if it helps us better compress the set of locations for these features. Conversely, a spatial correlation is good, if it helps us better compress the set of features in the corresponding region. Our approach is scalable for large datasets (both number of locations and of features).

Part III

Stream mining

Chapter 11

Introduction

Data streams have received considerable attention in various communities (theory, databases, data mining, networking, systems), due to several important applications, such as network analysis [CJSS03], sensor network monitoring [YG03], moving object tracking [Agg03], financial data analysis [ZS02], and scientific data processing [ZS03]. All these applications have in common that: (i) massive amounts of data arrive continuously, which makes traditional database systems prohibitively slow, and (ii) users, or higher-level applications, require immediate responses and cannot afford any post-processing (e.g., in network intrusion detection). Data stream systems have been prototyped [ACC$^+$03, MWA$^+$03, CCD$^+$03] and deployed in practice [CJSS03]. In addition to providing SQL-like support for data stream management systems (DSMS), it is crucial to detect patterns and correlations that may exist in data streams.

In this part we consider semi-infinite, time series data streams. Formally, such a stream is a discrete sequence of numbers $x_1, x_2, \ldots, x_t, \ldots$. Several applications produce huge amounts of data in this form [GKMS01, GKS01, DGGR02, GG02], where individual values are typically samples or measurements. Time sequences have attracted attention [BD91], for forecasting in financial, sales, environmental, ecological and biological time series, to mention a few. However, several new and exciting applications have recently become possible.

The emergence of cheap and small sensors has attracted significant attention. Sensors are small devices that gather measurements—for example, temperature readings, road traffic data, geological and astronomical observations, patient physiological data, etc. There are numerous, fascinating applications for such sensors and sensor networks, in fields such as health care and monitoring, industrial process control, civil infrastructure [CGN00], road traffic safety and smart houses, to mention a few. Although current small sensor prototypes [HSW$^+$00] have limited resources (512 bytes to 128Kb of storage), dime-sized devices

with memory and processing power equivalent to a PDA are not far away. In fact, PDA-like devices with data gathering units are already being employed in some of the above applications. The goal in the next decade is single-chip computers with powerful processors and 2–10Gb [CGN00] of non-volatile storage. Furthermore, embedded processors are becoming ubiquitous and their power has yet to be harnessed. A few examples of such applications are (a) intelligent (*active*) disks [RFGN00] that learn input traffic patterns and do appropriate prefetching and buffering, (b) intelligent routers that monitor data traffic and simplify network management.

We use the term "sensor" broadly, to refer to any embedded computing device with fairly limited processing, memory and (optionally) communication resources and which generates a semi-infinite sequence of measurements.

Stream data also arise naturally in several other important applications, such as network monitoring and financial applications. Detailed network usage and profiling data are collected in modern telecommunications networks. One of the main goals are performance monitoring and analysis, which in turn will help in improved network planning [GKMS01, LPC+04]. Furthermore, numerous network probes are deployed today in the Internet, to aid in anomaly and intrusion detection. Such data are collected from multiple locations into large repositories [ISC] and subsequently analysed.

In the financial domain, the canonical example is is stock quotes, which generate thousands of streams updated every few minutes [ZS02]. The analysis of market trends and correlations is a very important application. Furthermore, financial institutions collect and maintain account historical data, such as balance or transfer activity over time. Again, trends and correlations in such time series may help in planning as well as fraud detection. In all these cases, incremental processing of such data and any-time reporting of trends and correlations is an important but difficult task.

In Chapter 12 we focus on a *single* stream and focus on how to capture trends at multiple time scales using limited resources. Our proposed method, AWSOM, can do this automatically, i.e., with no prior inspection of the data or any user intervention and expert tuning before or during data gathering. Our algorithms require limited resources and can thus be incorporated in sensors—possibly alongside a distributed query processing engine [CCC+02, BGS01, MSHR02]. Updates are performed in constant time with respect to stream size, using logarithmic space. Existing forecasting methods (SARIMA, GARCH, etc) or "traditional" Fourier and wavelet analysis fall short on one or more of these requirements.

In Chapter 13 we examine *multiple* streams and consider the problem of capturing correlations and finding hidden variables corresponding to trends on collections of time series

streams. Our proposed method, SPIRIT, can incrementally find correlations and hidden variables, which summarise the key trends in the entire stream collection. It can do this quickly, with no buffering of stream values and without comparing pairs of streams. Moreover, it is any-time, single pass, and it dynamically detects changes. The discovered trends can also be used to immediately spot potential anomalies, to do efficient forecasting and, more generally, to dramatically simplify further data processing.

Chapter 12

Patterns on a single stream

The limitations on available memory and computational resources on sensors unavoidably imply the need for certain trade-offs—it is impossible to store everything. Furthermore, we want to make the most of available resources, allowing the sensor to adapt and operate without supervision for as long as possible. This is the problem we address in this chapter, for a *single* stream of numerical values. The goal is a "language" (i.e., model/representation) for efficient and effective stream mining. We want to collect information, in real-time and without any human intervention, and discover patterns such as those illustrated in Figure 12.1.

This problem is orthogonal to that of continuous query processing. We focus on an adaptive algorithm that can look for arbitrary patterns and requires no prior knowledge and initial human tuning to guide it. There are situations when we do not know *beforehand* what we are looking for. Furthermore, it may be impossible to guide the sensor as it collects data, due to the large volume of data and/or limited or unavailable communication. If further exploration is desired, users can issue further queries, guided by the general long-term patterns to quickly narrow down the "search space."

In detail, the main requirements are (see also Figure 12.1):

1. *Concise models:* The models should be able to capture most of the regularities in real-world signals, using limited resources. In particular, we want

   (a) Periodic component identification; humans can achieve this task, visually, from the time-plot. Our method should automatically spot multiple periodic components, each of unknown, arbitrary period.

   (b) Noise filtering/identification; various types of "noise" are present in most real

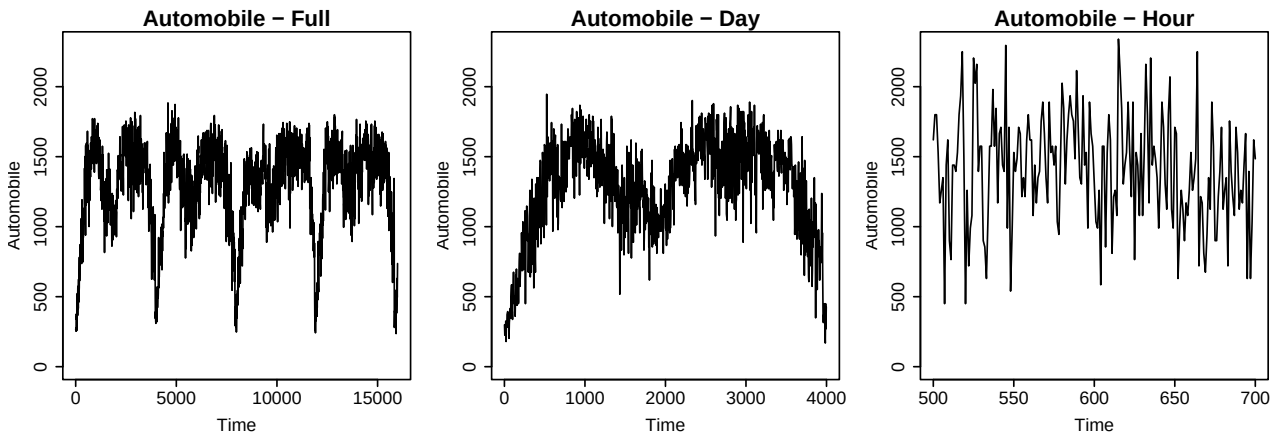


Figure 12.1: Automobile traffic, complete series, one day and one hour (the first two are 8- and 4-point averages to make the trends easier to see). There is clearly a daily periodicity. Also, in each day there is another distinct pattern (morning and afternoon rush hours). However, at an hour scale traffic is highly bursty—in fact, it can be modelled by *self-similar* noise. We want a method that can capture all this information automatically, with one pass and using limited memory!

signals; our framework should deal with it and identify several of its characteristics.

(c) Finally, we need a few, simple patterns (i.e., equations and features), so they can be easily communicated to other nearby sensors and/or to a central processing site.

2. *Streaming framework:* In particular, we want

   (a) An online, one-pass algorithm, since we can afford neither the memory nor time for offline updates, much less multiple passes over the data stream

   (b) Limited memory use, since sensor memory will be exhausted, unless our method carefully detects redundancies (or equivalently, patterns) and exploits them

   (c) Any-time forecasting and outlier detection; it is not enough to do compression (e.g., of long silence periods, or by ignoring small Fourier or wavelet coefficients). The model should be *generative* and thus able to report outliers. An outlier can be defined as any value that deviates too much from our forecast (e.g., by two standard deviations).

3. *Unsupervised, automatic operation:* In a general sensor setting, we cannot afford human intervention.

The AWSOM framework can extract several key features of the stream, in a principled way. It can do so in a single pass, with minimal resource requirements. Based on these features, it can immediately provide information about the stream at several levels. In brief, AWSOM has all of the above characteristics, while none of the previously published methods (AR and variations, Fourier analysis, wavelet decomposition—see Section 12.1.2) can claim the same.

12.1 Related work

Previous work broadly falls into two categories. The first includes work done by the databases community on continuous query processing. These methods employ increasingly sophisticated mathematical methods, but the focus is typically on some form of compression (or, *synopses*) which do not employ generative models. The other includes various popular statistical models and methods for time series forecasting. However, standard estimation methods for these models typically fail in one or both of the requirements (2) and (3) earlier.

Thus the authors believe that there is a need for straightforward methods of time series model building which can be applied in real-time to semi-infinite streams of data, using limited memory.

12.1.1 Continuous queries and stream processing

An interesting method for discovering *representative trends* in time series using *sketches* was proposed by Indyk et al. [IKM00]. A representative trend is a section of the time series that has the smallest sum of "distances" from *all* other sections of the same length. The proposed method employs random projections for dimensionality reduction and FFT to quickly compute the sum of distances. However, it cannot be applied to semi-infinite streams, since each section has to be compared to every other.

Related to the above is the very recent approach of Ergün et al. [EMS04]. They define *approximate periodicity* based on self-distances (extending the notion of exact periodicity from combinatorial pattern matching) and present efficient sampling techniques that require only $O(\sqrt{n})$ samples to compute self-distances for any potential period, without any prior knowledge.

Gilbert et al. [GKMS01] use wavelets to compress the data into a fixed amount of memory, by keeping track of the largest Haar wavelet coefficients and carefully updating them on-line. In the following, we will use the name *Incremental DWT* or *IncDWT* for short. How-

ever, this method does not try to discover patterns and trends in the data. Thus, it cannot compete directly with our method, which employs a *generative* model. More recently, Garofalakis et al. [GG02] presented an approach for accurate data compression using *probabilistic wavelet synopses*. However, this method has an entirely different focus and cannot be applied to semi-infinite streams. The recent work of Guha et al. [GK02] efficiently constructs ϵ-accurate histograms in a stream setting, for a fixed number of buckets, using space and time-per-element that is logarithmic with respect to stream size (or poly-logarithmic with respect to a fixed window size). Further work on streams focuses on providing exact answers to pre-specified sets of queries using the minimum amount of memory possible. Arvind et al. [ABB+02] study the memory requirements of continuous queries over *relational* data streams. Datar et al. [DGI+02] keep *exact* summary statistics and provide theoretical bounds in the setting of a bit stream. Das et al. [DLM+98] examine the problem of discovering rules in time series, by quantising them and then mining local, frequent-itemset type rules over sliding window fragments, based on certain similarity criteria.

There is also recent work on approximate answers to various types of continuous queries. Gehrke et al. [GKS01] presents a comprehensive approach for answering correlated aggregate queries (e.g., "find points below the (current) average"), using histogram "summaries" to approximate aggregates. Dobra et al. [DGGR02] present a method for approximate answers to aggregate multi-join queries over several streams, using random projections and boosting. More recently, Considine et al. [CLKB04] have proposed novel sketching techniques for aggregate queries with the goal of minimising communication and computation overhead.

Olston et al. [OJW03] present a query processing framework for minimising communication overhead over a set of specific continuous queries over multiple streams, while providing error guarantees. Also, the work in [BO03] considers the problem of efficiently monitoring the top-k values in a distributed, multiple stream setting.

Zhu and Shasha [ZS03] examine the problem of efficient detection of *elastic bursts* in streams. In [ZS02] they use the DFT to summarise streams within a finite window and then compute the pairwise correlations among all streams. Also, Yi et al. [YSJ+00] present a method for analysis of multiple co-evolving sequences.

A system for linear pattern discovery on multi-dimensional time series was presented recently by Chen et al. [CDH+02]. Although this framework employs varying resolutions *in time*, it does so by straight aggregation, using manually selected aggregation levels (although the authors discuss the use of a geometric progression of time frames) and can only deal with, essentially, linear trends. Recently, Bulut et al. [BS03b] proposed an approach

Method	Contin. Streams	Trends / Forecast	Auto-matic	Memory
DFT (N-point)	NO	NO	—	—
SWFT (N-point)	YES(?)	NO	—	—
DWT (N-point)	NO	NO	—	—
IncDWT [GKMS01]	YES	NO	—	—
Sketches [IKM00]	NO	YES(?)	—	—
AR / ARIMA	YES	YES	NO [BD91]	W^2
AWSOM	YES	YES	YES	$m\|\mathcal{D}\|^2$

Table 12.1: Comparison of methods.

for hierarchical stream summarisation (similar to that of [GKMS01]) which focuses on simple queries and communication/caching issues for wavelet coefficients. Even more recently, Palpanas et al. [PVK+04] consider approximation of time-series with *amnesic* functions. They propose novel techniques suitable for streaming, and applicable to a wide range of user-specified approximating functions.

Finally, in other areas of database research, Zhang et al. [ZGTS02] present a framework for spatio-temporal joins using multiple-granularity indices. Aggregation levels are pre-specified and the main focus is on efficient indexing. Tao et al [TFPL04] recently presented an approach to recursively predict motion sequence patterns.

12.1.2 Time series methods

None of the continuous querying methods deal with pattern discovery and forecasting. The typical "textbook" approaches to forecasting (i.e., generative time series models) include *auto-regressive (AR)* models and their generalisations, *auto-regressive moving average (ARMA)*, *auto-regressive integrated moving average (ARIMA)* and *seasonal ARIMA (SARIMA)* [BD91]. Other popular time-series models include *GARCH (generalised auto-regressive conditional heteroskedasticity)* [Bol86] and *ARFIMA (auto-regressive fractionally integrated moving average)* [Ber94]. These time-series models could be used; however, standard estimation methods for these models fail in one or both of the requirements (2) and (3) stated in the introduction. Furthermore, these methods often have a number of other limitations.

Existing model-fitting methods are typically batch-based (i.e., do not allow online update of parameters). Established methods for determining model structure are at best computationally intensive, besides not easily automated. Large window sizes introduce severe estimation problems, both in terms of resource requirements as well as accuracy.

Symbol	Definition
$\mathbf{X}, \mathbf{P}, \ldots$	Matrices (boldface capital).
$\mathbf{y}, \mathbf{q}, \mathbf{b}, \ldots$	Vectors (boldface lower-case).
X_t	Value at time $t = 0, 1, \ldots$ (sometimes also $X[t]$).
N	Number of points *so far* from $\{X_t\}$.
$W_{l,t}$	Wavelet (or, detail) coefficient (level l, time t); also denoted $W[l, t]$
$V_{l,t}$	Scaling (or, smooth) coefficient (level l, time t); also denoted $V[l, t]$.
$\beta_i^{(l)}$	AWSOM coefficient at time lag i, for the equation at level l.
AWSOM(n_0)	AWSOM model of order n_0 (i.e., with n_0 model coefficients, per level). The *total order* of this model is $k \equiv n_0$.

Table 12.2: Symbols and definitions.

In addition, ARIMA models cannot handle bursty time series, even when the bursts are re-occurring. While GARCH models [Bol86] can handle the class of bursty *white noise* sequences, they do not have the richness in structure to model a wide variety of different time series. Recently, the ARIMA model has been extended to ARFIMA, which handles the class of *self-similar* bursty sequences [Ber94]. However, ARFIMA models pose particular computational problems since they cannot be expressed in Markovian form, making the likelihood computationally burdensome to evaluate.

All the above methods deal with linear forecasting. Non-linear modelling methods [WG94] also require human intervention to choose the appropriate windows for non-linear regression or to configure an artificial neural network.

12.1.3 Other

There is a large body of work in the signal processing literature related to compression and feature extraction. Typical tools include the Fast Fourier Transform (FFT), as well as the Discrete Wavelet Transform (DWT) [PW00]. However, most of these algorithms (a) deal with *fixed length* signals of size N, and (b) cannot do forecasting (i.e., do not employ a *generative* model).

12.2 Background material

In this section we give a very brief introduction to some necessary background material.

12.2.1 Auto-regressive (AR) modelling

Auto-regressive models are the most widely known and used. We present the basic ideas—more information can be found in, e.g., [BD91]. The main idea is to express X_t as a function of its previous values, plus (filtered) noise ϵ_t:

$$X_t \;=\; \phi_1 X_{t-1} \;+\; \ldots \;+\; \phi_W X_{t-W} \;+\epsilon_t \tag{12.1}$$

where W is a window that is determined by trial and error, or by using a criterion that penalises model complexity (i.e., large values of W), like the *Akaike Information Criterion (AIC)*. Seasonal variants (SAR, SAR(I)MA) also use window offsets that are multiples of a single, fixed period (i.e., besides terms of the form X_{t-i}, the equation contains terms of the form X_{t-Si} where S is a constant). The typical ARIMA modelling approach involves manual preprocessing to remove trend and seasonal components.

In more detail, an auto-regressive model of order p, or AR(p) express X_t as a linear combination of previous values, i.e., $X_t = \phi_1 X_{t-1} + \cdots + \phi_p X_{t-p} + \epsilon_t$ or, more concisely

$$\phi(L) X_t = \epsilon_t$$

where L is the lag operator and $\phi(L)$ is a polynomial defined on this operator:

$$\begin{aligned} LX_t &\equiv X_{t-1} \\ \phi(L) &= 1 - \phi_1 L - \phi_2 L^2 - \cdots - \phi_p L^p \end{aligned}$$

and ϵ_t is a white noise process, i.e.,

$$\mathrm{E}[\epsilon_t] = 0 \quad \text{and} \quad \mathrm{Cov}[\epsilon_t, \epsilon_{t-k}] = \begin{cases} \sigma^2 & \text{if } k = 0 \\ 0 & \text{otherwise} \end{cases}$$

Using least-squares, we can estimate σ^2 from the sum of squared residuals (SSR). This is used as a measure of estimation error; when generating "future" points, ϵ_t is set to $\mathrm{E}[\epsilon_t] \equiv 0$.

The next step up are auto-regressive moving average models. An ARMA(p, q) model expresses values X_t as

$$\phi(L) X_t = \theta(L) \epsilon_t$$

where $\theta(L) = 1 - \theta_1 L - \cdots - \theta_q L^q$. Estimating the moving average coefficients θ_i is fairly involved. State of the art methods use maximum-likelihood (ML) algorithms, employing

iterative methods for non-linear optimisation, whose computational complexity depends exponentially on q.

ARIMA(p, d, q) models are similar to ARMA(p, q) models, but operate on $(1 - L)^d X_t$, i.e., the d-th order backward difference of X_t:

$$\phi(L)(1 - L)^d X_t = \theta(L)\epsilon_t$$

Finally, SARIMA$(p, d, q) \times (P, D, Q)T$ models are used to deal with seasonalities, where:

$$\phi(L)\Phi(L^T)(1 - L)^d(1 - L^T)^D X_t = \theta(L)\Theta(L^T)\epsilon_t$$

where the seasonal difference polynomials

$$\Phi(L^T) = 1 - \Phi_1 L^T - \Phi_2 L^{2T} - \cdots - \Phi_P L^{PT}$$
$$\Theta(L^T) = 1 - \Theta_1 L^T - \Theta_2 L^{2T} - \cdots - \Theta_Q L^{QT}$$

are similar to $\phi(L)$ and $\theta(L)$ but operate on lags that are multiples of a fixed period T. The value of T is yet another parameter that either needs to be estimated or set based on prior knowledge about the series X_t.

12.2.2 ACF and PACF

The *auto-correlation function (ACF)* and *partial auto-correlation function (PACF)* are traditionally used to gain an understanding of time-series behaviour.

Definition 21 (ACF). *For a (weakly) stationary series* $\{X_t\}$*, the* autocovariance function *is defined as*

$$\gamma_k := \mathrm{Cov}[X_t - \mu, X_{t-k} - \mu] \equiv \mathrm{E}\big[(X_t - \mu)(X_{t-k} - \mu)\big],$$

where $\mu := \mathrm{E}[X_t]$*. The* autocorrelation function (ACF) *then is*

$$\rho_k := \gamma_k / \gamma_0,$$

i.e., the correlation coefficients at lag k.

By definition, $\rho_0 = 1$. Also, assuming a white noise process, approximate 95% confidence intervals are given by $\pm 1.96\sqrt{N}$, where N is the number of points in the series.

For a pure white noise process, the ACF is zero for all values ($\rho_k = 0$, for all $k \geq 1$). For a

pure auto-regressive (AR) process, the ACF decays exponentially. Finally, for a pure moving average process of order q (MA(q)), the ACF is non-zero for values up to q ($\rho_k = 0$ iff $k > q$).

In this section, we examine the ACF per DWT level. Since we know that, by construction $\mathrm{E}_t[W_{l,t}] = 0$ for every l, the autocovariances and autocorrelations are

$$\gamma_k^{(l)} := \mathrm{Cov}_t[W_{l,t}, W_{l,t-k}] \quad \text{and} \quad \rho_k^{(l)} := \gamma_k^{(l)} / \gamma_0^{(l)}.$$

Traditionally, when studying auto-regressive processes, the partial autocorrelation function is also used.

Definition 22 (PACF). *The* partial autocorrelation function (PACF) α_k *at lag* k *is the correlation coefficient between between the two sets of residuals obtained by regressing* X_t *and* X_{t-k} *on the intervening values* $X_{t-k+1}, \ldots, X_{t-1}$.

In particular, $\alpha_1 = \rho_1$ and α_k ($k > 1$) is a measure of the correlation between X_t and X_{t-k} after the (linear) effect of intervening observations has been removed. The PACF α_k at lag k is typically estimated by fitting an AR(k) model to the data.

Therefore, for a pure AR(p) process, the PACF is non-zero for values up to p ($\alpha_k = 0$ iff $k > p$). For a pure MA process, the PACF decays exponentially.

In practice, real series are not so "clear cut" with respect to the ACF and PACF and things are complicated somewhat further by sampling effects when dealing with real data. Thus, this information must be interpreted with some care, but it still provides guidance.

12.2.3 Recursive Least Squares (RLS)

Recursive Least Squares (*RLS*) is a method that allows dynamic update of a least-squares fit. The least squares solution to an overdetermined system of equations $\mathbf{Xb} = \mathbf{y}$ where $\mathbf{X} \in \mathbb{R}^{m \times k}$ (measurements), $\mathbf{y} \in \mathbb{R}^m$ (output variables) and $\mathbf{b} \in \mathbb{R}^k$ (regression coefficients to be estimated) is given by the solution of $\mathbf{X}^T\mathbf{Xb} = \mathbf{X}^T\mathbf{y}$. Thus, all we need for the solution are the projections

$$\mathbf{P} \equiv \mathbf{X}^T\mathbf{X} \qquad \text{and} \qquad \mathbf{q} \equiv \mathbf{X}^T\mathbf{y}$$

We need only space $O(k^2 + k) = O(k^2)$ to keep the model up to date. When a new row $\mathbf{x}_{m+1} \in \mathbb{R}^k$ and output y_{m+1} arrive, we can update

$$\mathbf{P} \leftarrow \mathbf{P} + \mathbf{x}_{m+1}\mathbf{x}_{m+1}^T \quad \text{and}$$
$$\mathbf{q} \leftarrow \mathbf{q} + y_{m+1}\mathbf{x}_{m+1}.$$

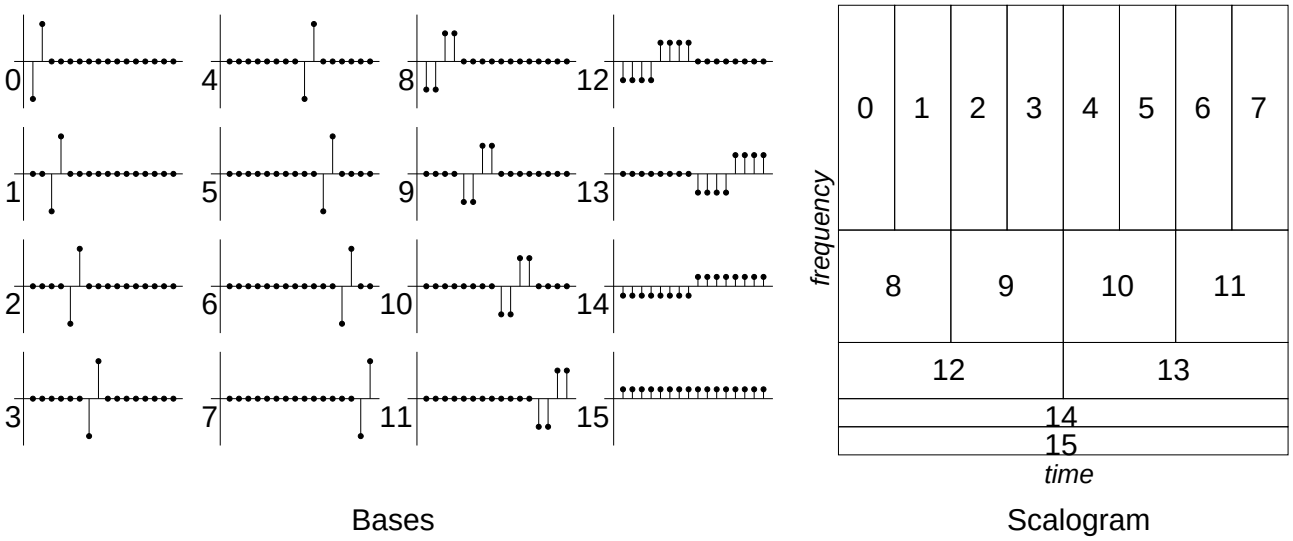


Figure 12.2: Haar bases and correspondence to time/frequency (for signal length $N = 16$). Each wavelet coefficient is a linear projection of the signal to the respective basis.

In fact, it is possible to update the regression coefficient vector $\mathbf{b}$ without explicitly inverting $\mathbf{P}$ to solve $\mathbf{b} = \mathbf{P}^{-1}\mathbf{q}$. In particular (see, e.g., [You84]) the update equations are

$$\mathbf{G} \leftarrow \mathbf{G} - (1 + \mathbf{x}_{m+1}^T \mathbf{G} \mathbf{x}_{m+1})^{-1} \mathbf{G} \mathbf{x}_{m+1} \mathbf{x}_{m+1}^T \mathbf{G}$$
$$\mathbf{b} \leftarrow \mathbf{b} - \mathbf{G} \mathbf{x}_{m+1} (\mathbf{x}_{m+1}^T \mathbf{b} - y_{m+1}),$$

where the matrix $\mathbf{G}$ can be initialised to $\mathbf{G} \leftarrow \epsilon \mathbf{I}$ (where ϵ is a small positive number and $\mathbf{I}$ is the $k \times k$ identity matrix).

RLS and AR In the context of auto-regressive modelling (Eq. 12.1), we have one equation for each stream value $X_{w+1}, \ldots, X_t, \ldots$, i.e., the m-th row of the $\mathbf{X}$ matrix above is

$$\mathbf{x}_m = [X_{m-1} \; X_{m-2} \; \cdots \; X_{m-w}]^T \in \mathbb{R}^w$$

for $t - w = m = 1, 2, \ldots$ ($t > w$). In this case, the solution vector $\mathbf{b}$ consists precisely of the auto-regression coefficients in Eq. 12.1, i.e.,

$$\mathbf{b} = [\phi_1 \; \phi_2 \; \cdots \; \phi_w]^T \in \mathbb{R}^w.$$

12.2.4 Wavelets

The N-point *discrete wavelet transform (DWT)* of a length N time sequence gives N wavelet coefficients. Wavelets are best introduced with the Haar transform, because of its simplicity (a more rigorous introduction can be found, e.g., in [PW00]). At each level l of the construction we keep track of two sets of coefficients, each of which "looks" at a time window of size 2^l:

- $V_{l,t}$: The *smooth* component, which consists of the $N/2^l$ *scaling coefficients*. These capture the low-frequency component of the signal; in particular, the frequency range $[0, 1/2^l]$.
- $W_{l,t}$: The *detail* component, which consists of the $N/2^l$ *wavelet coefficients*. These capture the high-frequency component; in particular, the range $[1/2^l, 1/2^{l-1}]$.

The construction starts with $V_{0,t} = X_t$ and $W_{0,t}$ is not defined. At each iteration $l = 1, 2, \ldots, \lg N$ we perform two operations on $V_{l-1,t}$ to compute the coefficients at the next level:

- Differencing, to extract the high frequencies:

$$W_{l,t} = (V_{l-1,2t} - V_{l-1,2t-1})/\sqrt{2}$$

- Smoothing, which averages[1] each consecutive pair of values and extracts the low frequencies:

$$V_{l,t} = (V_{l-1,2t} + V_{l-1,2t-1})/\sqrt{2}$$

We stop when $W_{l,t}$ consists of one coefficient (which happens at $l = \lg N + 1$). The scaling coefficients are needed only during the intermediate stages of the computation. The final wavelet transform is the set of all wavelet coefficients along with $V_{\lg N+1,0}$. Starting with $V_{\lg N+1,0}$ (which is also referred to as the signal's scaling coefficient) and following the inverse steps, we can reconstruct each $V_{l,t}$ until we reach $V_{0,t} \equiv X_t$.

Figure 12.2 illustrates the final effect for a signal with $N = 16$ values. Each wavelet coefficient is the result of projecting the original signal onto the corresponding basis signal (i.e., taking the dot product of the signal with the basis). Figure 12.2 shows the *scalogram*, that is, the energy (i.e., squared magnitude) of each wavelet coefficient versus the location in time and frequency it is "responsible" for. In general, there are many wavelet transforms, but they all follow the pattern above: a wavelet transform uses a pair of filters, one high-pass and one

[1] The scaling factor of $1/\sqrt{2}$ in both the difference and averaging operations is present in order to preserve total signal energy (i.e., sum of squares of all values).

low-pass. For example, in Haar wavelets, this pair consists of the simple differencing and averaging filters, respectively.

For our purposes here, we shall restrict ourselves to wavelets of the Daubechies family, which have desirable smoothness properties and successfully compress many real signals. In practice, although by far the most commonly used (largely due to their simplicity), Haar wavelets are too unsmooth and introduce significant artifacting [PW00]. In fact, unless otherwise specified, we use Daubechies-6.

Incremental wavelets This part is a very brief overview of how to compute the DWT incrementally. This is the main idea of IncDWT [GKMS01], which uses Haar wavelets. In general, when using a wavelet filter of length L, the wavelet coefficient at a particular level is computed using the L corresponding scaling coefficients of the previous level. Recall that $L = 2$ for Haar (average and difference of two consecutive points), and $L = 6$ for Daubechies-6 that we typically use. Thus, we need to remember the last $L - 1$ scaling coefficients at each level. We call these the *wavelet crest*.

Definition 23 (Wavelet crest). *The* wavelet crest *at time t is defined as the set of scaling coefficients (wavelet smooths) that need to be kept in order to compute the new wavelet coefficients when X_t arrives.*

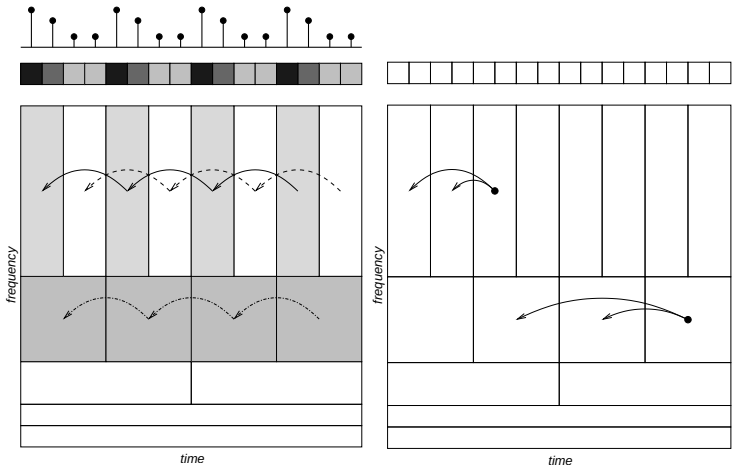
Lemma 9 (DWT update). *Updating the wavelet crest requires space $(L-1)\lg N + L = O(L \lg N) = O(\lg N)$, where L is the width of the wavelet filter (fixed) and N the number of values seen so far.*

Proof. See [GKMS01]. Generalising to non-Haar wavelets and taking into account the wavelet filter width is straightforward. □

Wavelet properties

In this section we emphasise the DWT properties which are relevant to AWSOM.

Computational complexity The DWT can be computed in $O(N)$ time and, as new points arrive, it can be updated in $O(1)$ amortised time. This is made possible by the structure of the time/frequency decomposition which is unique to wavelets. For instance, the Fourier transform also decomposes a signal into frequencies (i.e., sum of sines), but requires $O(N \lg N)$ time to compute and cannot be updated as new points arrive.



(a) (b)

Figure 12.3: AWSOM—Intuition and demonstration. AWSOM captures intra-scale correlations (a). Also, (b) demonstrates why we fit different models per level.

Time/frequency decomposition Notice (see scalogram in Figure 12.2) that higher level coefficients are highly localised in time, but involve uncertainty in frequency and vice-versa. This is a *fundamental* trade-off of any time/frequency representation and is a manifestation of the *uncertainty principle*, according to which localisation in frequencies is inversely proportional to localisation in time. The wavelet representation is an excellent choice when dealing with semi-infinite streams in limited memory: it "compresses" well many real signals, while it is fast to compute and can be updated online.

Wavelets and decorrelation A wavelet transform with filter of length $2L$ can decorrelate only certain signals provided their L-th order (or less) backward difference [2] is a stationary random process [PW00]. For real signals, this value of L is not known in advance and may be impractically large: the space complexity of computing new wavelet coefficients is $O(L \lg N)$—see Lemma 9.

For a more detailed analysis and discussion of the properties of real-world signals, see Section 12.5.2.

Wavelet variance One further benefit of using wavelets is that they decompose the variance across scales. Furthermore, the plot of log-power versus scale can be used to detect self-similar components (see Section 12.2.5 for a brief overview).

[2] In particular, the Daubechies-$2L$ wavelet filters are essentially L-th order backward differences (and, consequently, the scaling filters are essentially $2L$-point weighted moving averages).

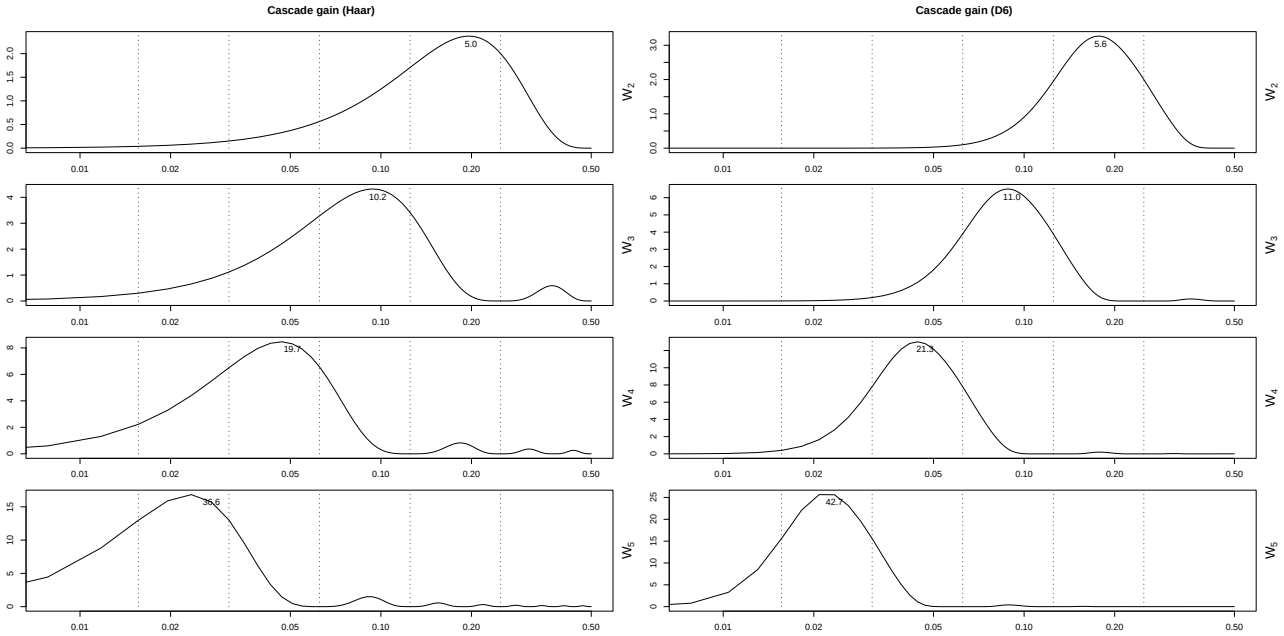


Figure 12.4: Illustration of Haar and Daubechies-6 cascade gain (levels 3–5). The horizontal axis is frequency and the curves show how much of each frequency is "represented" at each wavelet level. As expected, D-6 filters (used in all experiments), have better band-pass properties.

12.2.5 More wavelet properties

Frequency properties Wavelet filters employed in practice can only approximate an ideal bandpass filter, since they are of *finite* length L. The practical implications are that wavelet coefficients at level l correspond roughly to the frequency range $[1/2^{l+1}, 1/2^l]$ (or, equivalently, periods in $[2^l, 2^{l+1}]$ (see Figure 12.4 for the actual correspondence). This has to be taken into account for precise interpretation of AWSOM models by an expert.

Wavelet variance and self-similarity The wavelet variance decomposes the variance of a sequence across scales. Due to space limitations, we mention basic definitions and facts; details can be found in [PW00].

Definition 24 (Wavelet variance). *If $\{W_{l,t}\}$ is the DWT of a series $\{X_t\}$ then the wavelet variance $\mathcal{V}_l$ is defined as $\mathcal{V}_l = \mathrm{Var}[W_{l,t}]$*

Under certain general conditions, $\hat{\mathcal{V}}_l = \frac{2^l}{N} \sum_{t=1}^{N/2^l} W_{l,t}^2$ is an *unbiased* estimator of $\mathcal{V}_l$. Note that the sum is precisely the energy of $\{X_t\}$ at scale l.

Definition 25 (Self-similar sequence). *A sequence $\{X_t\}$ is said to be self-similar following a pure power-law process if $\mathcal{S}_X(f) \propto |f|^\alpha$, where $-1 < \alpha < 0$ and $\mathcal{S}_X(f)$ is the SDF*[3]

It can be shown that $\mathcal{V}_l \approx 2 \int_{1/2^{l+1}}^{1/2^l} \mathcal{S}_X(f) df$, thus if $\{X_t\}$ is self-similar, then

$$\log \mathcal{V}_l \propto l,$$

i.e., the plot of $\log \mathcal{V}_l$ versus the level l should be linear. In fact, slope of the log-power versus scale plot should be approximately equal to the exponent α. This fact and how to estimate $\mathcal{V}_l$ are what the reader needs to keep in mind.

—-

12.3 Proposed method

In this section we introduce our proposed model. What equations should we be looking for to replace ARIMA's (see Equation 12.1)?

12.3.1 Intuition behind our method

First part—information representation This is a crucial choice—what is a good way to represent the key information in the series, given the severe resource constraints in a streaming, sensor setting? We want a powerful and flexible representation that can adapt to the sequence, rather than expect someone to adapt the sequence to the representation. We propose to use wavelets because they are extremely successful in compressing most real signals, such as voice and images [Fal96], seismic data [ZdZ98], biomedical signals [Aka97] and economic time sequences [GSW01]. By using wavelet coefficients, we immediately discard many redundancies (i.e., near-zero valued wavelet coefficients) and focus on what really matters. Furthermore, the DWT can be computed quickly and updated online.

Second part—correlations In the wavelet domain, how can we capture arbitrary periodicities? A periodic signal will have high-energy wavelet coefficients at the scales that correspond to its frequency. Also, successive coefficients on the same level should have related

[3]The *spectral density function (SDF)* is the Fourier transform of the auto-covariance sequence (ACVS) $S_{X,k} \equiv \text{Cov}[X_t, X_{t-k}]$. Intuitively, it decomposes the variance into frequencies.

values (see Figure 12.3(a)). Thus, in order to capture periodic components, we should look for intra-scale correlations between wavelet coefficients[4].

The last question we need to answer is: what type of regression models should we use to quantify these correlations? Our proposed method tries to capture these by fitting linear regression models in the wavelet domain. These can also be updated online with RLS.

Summary

We propose using the wavelet representation of the series and capturing correlations in the wavelet domain (see Figure 12.3(b)). If we naïvely try to apply linear auto-regression in the time domain, there are several problems:

- Window size: In order to capture long-term (non-sinusoidal) periodic components, the window size has to be as large as the period. If we hope to capture *any* information about long-term periodic trends, the window size has to be extremely large and this clearly violates the limited memory requirements (it also creates other problems, as discussed next).

  However, the wavelet coefficients at each level capture information at a coarser resolution, but at windows whose size increases exponentially with the level.

- Noise: Even with a large window size, the presence of noise (typically at higher frequencies) severely affects model fitting. Dealing with noise in the time domain is possible (moving average being the simplest approach), but cannot be done easily in an online setting and often requires human intervention (seasonal models typically used for this reason, where the period has to be manually identified).

  However, the wavelet transform filters out this noise quite successfully and does so in a principled manner, than has been proven to work for several real signals. Furthermore, it allows us to extract certain important characteristics of the noise.

Therefore, our approach significantly improves modelling power while dramatically reducing memory requirements, by essentially performing auto-regression on the wavelet representation instead of the original signal. At the same time, it adheres to the principle of no prior human intervention and tuning (or, adapting to new data as they arrive).

[4]Bursts, on the other hand, carry energy in most frequencies and give rise to inter-scale correlations; see also the discussion in Section 12.5.1 for more details on this aspect.

Input: Continuous, numerical stream $X[1], X[2], \ldots, X[t], \ldots$ **Where:** N is the number of points seen so far
Model order n_0 (typically $n_0 \leq 6$) $1 \leq l \leq \lceil \lg N \rceil$ and $1 \leq i \leq n_0$
Output: Wavelet variances $\mathcal{V}_l$
AWSOM coefficients $\beta_i^{(l)}$
Wavelet crest (or, initial conditions) $W[N-i-1, l]$

UpdateCrest ($X[t]$):
Foreach $l \geq 0$ s.t. 2^l divides t:
Compute $V[l, t/2^l]$
If 2^{l+1} divides t:
Compute $W[l, t/2^{l+1}]$
Delete $W[l, t/2^{l+1} - L]$

Update ($X[t]$):
UpdateCrest($X[t]$)
Foreach new coefficient $W[l, t']$ in the crest:
Find the linear model it belongs to
based on l and $t' \mod \Lambda$
Update $\mathbf{P} \equiv \mathbf{X}^T\mathbf{X}$ and
$\mathbf{q} \equiv \mathbf{X}^T\mathbf{y}$ for this model

ModelSelection:
For each linear model:
Estimate SSR of complete model
For each subset of regression variables:
Compute SSR of reduced model
from 10
Estimate probability that reduction
in variance is not due to chance
Select the subset of variables with
highest probability (or keep all
if not within 95% confidence interval)

Figure 12.5: High-level description of the algorithms.

12.3.2 AWSOM modelling

We express the wavelet coefficients at each level as a function of the n_0 previous coefficients of the same level, i.e.,

$$W_{l,t} = \beta_1^{(l)} W_{l,t-1} + \beta_2^{(l)} W_{l,t-2} + \cdots + \beta_{n_0}^{(l)} W_{l,t-n_0} \tag{12.2}$$

where $W_{l,t}$ are the wavelet coefficients (at time t and level l) and $\beta_i^{(l)}$ are the AWSOM coefficients (for level l and lag i). We estimate one set of such coefficients for *each* level l; note that l is *at most* $\lg N$. This is a model of order n_0, denoted as AWSOM(n_0). To summarise:

Definition 26 (AWSOM **model and order).** *A model with n_0 coefficients $\beta_i^{(l)}$, $i = 1, 2, \ldots, n_0$ for each wavelet level l is denoted as* AWSOM(n_0). *Its* (total) order *is n_0.*

This can capture arbitrary periodic components and is sufficient in many real signals.

Figure 12.5 described the algorithm; we update the wavelet crest on-line (as described before) and use recursive least squares to update the regression models, also online.

12.3.3 Model selection

Many of the dependencies may be statistically insignificant, so the respective coefficients $\beta(l)_i$ should be set to zero. We want to (a) avoid over-fitting and (b) present to the user those patterns that are important. We can do model selection using *only* data gathered online and with time complexity *independent* of the stream size. Next, we show how feature selection can be done from the data gathered online (i.e., P and $\mathbf{q}$ for each AWSOM equation). The

algorithm is sketched in Figure 12.5. The main idea is to determine whether the reduction in error achieved by adding extra parameters is statistically significant (based on some criterion), i.e., it cannot be attributed to noise.

Model testing and selection

The key quantity we need is the total squared error (or, *square sum of residuals (SSR)*).

Lemma 10 (Square sum of residuals). *If* $\mathbf{b}$ *is the least-squares solution to the overdetermined equation* $\mathbf{Xb} = \mathbf{y}$*, then*

$$s_n \equiv \sum_{i=1}^{n} (\mathbf{x_i}^T \mathbf{b} - y_i)^2 = \mathbf{b}^T \mathbf{P} \mathbf{b} - 2\mathbf{b}^T \mathbf{q} + \mathbf{y}^2$$

Proof. Straightforward from the definition of s_n, which in matrix form is $s_n = (\mathbf{Xb} - \mathbf{y})^2$. □

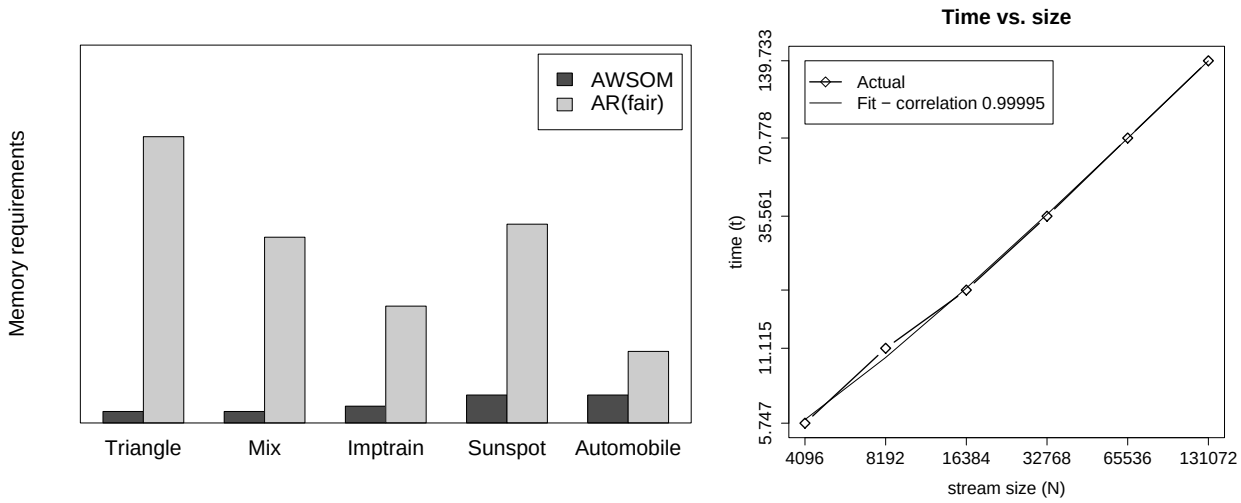
Thus, besides $\mathbf{P}$ and $\mathbf{q}$, we only need to update $\mathbf{y}^2$ (a single number), by adding y_i^2 to it as each new value arrives. Now, if we select a subset $\mathcal{I} = \{i_1, i_2, \ldots, i_p\} \subseteq \{1, 2, \ldots, k\}$ of the k variables $x_1, x_2, \ldots, x_k$, then the solution $\mathbf{b}_\mathcal{I}$ for this subset is given by $\mathbf{P}_\mathcal{I} \mathbf{b}_\mathcal{I} = \mathbf{q}_\mathcal{I}$ and the SSR by $s_n = \mathbf{b}_\mathcal{I}^T \mathbf{P}_\mathcal{I} \mathbf{b}_\mathcal{I} - 2\mathbf{b}_\mathcal{I} \mathbf{q}_\mathcal{I} + \mathbf{y}^2$ where the subscript $\mathcal{I}$ denotes straight row/column selection (e.g., $\mathbf{P}_\mathcal{I} = [p_{i_j, i_k}]_{i_j, i_k \in \mathcal{I}}$)

The *F-test (Fisher test)* [DS02] is a standard method for determining whether a reduction in variance is statistically significant. The F-test is based on the sample variances, which can be computed *directly* from the SSR (Lemma 10). Although the F-test holds precisely (i.e., non-asymptotically) under normality assumptions, in practice it works well in several circumstances, especially when the population size is large (as is the case with semi-infinite streams).

12.3.4 Complexity

In this section we show that our proposed AWSOM models can be easily estimated with a single-pass, "any-time" algorithm. From Lemma 9, estimating the new wavelet coefficients requires space $O(\lg N)$. In fact, since we typically use Daubechies-6 wavelets ($L = 6$), we need to keep exactly $5 \lg N + 6$ values. The AWSOM models can be dynamically updated using RLS.

At each level, we fit a separate regression model; since the number of levels is $\lceil \lg N \rceil$, this leads to the following result:



(a) (b)

Figure 12.6: (a) Memory space requirements (normalised): Space needed to keep the models up-to-date (AWSOM and AR with equivalent, fair window size. (b) Time complexity versus stream size (Python prototype), including model selection; the relationship is exactly linear, as expected.

Lemma 11 (Logarithmic space complexity). *Maintaining an* AWSOM(k) *model requires* $O(\lg N + mk^2)$ *space, where* N *is the length of the signal so far,* k *is the total* AWSOM *order and* $m = \lceil \lg N \rceil = O(\lg N)$ *the number of equations.*

Proof. Keeping the wavelet crest scaling coefficients requires space $O(\lg N)$. If we use recursive least squares, we need to maintain a $k \times k$ matrix for each of the m equations in the model. □

Auto-regressive models with a comparable window size need space $O(m^2k^2)$, since the equivalent fair window size is $W \approx mk$. Here, "fair" means that the number of total number of AWSOM coefficients plus the number of initial conditions we need to store is the same for both methods. This is the information that comprises the data synopsis and that would have to be eventually communicated. However, the device gathering the measurements needs extra storage space in order to update the models. The latter is, in fact, *much* larger for AR than for AWSOM (see Figure 12.6(a)). Thus this definition of equivalent window actually favours AR.

Lemma 12 (Time complexity). *Updating the model when a new data point arrives requires* $O(k^2)$ *time on average, where* k *is the number of* AWSOM *coefficients in each equation.*

Dataset	Size	Description
`Triangle`	64K	Triangle wave (i.e., piecewise linear; amplitude 5, period 256)
`Mix`	256K	Square wave (ampl. 25, period 256) plus sine (amplitude 5, period 64)
`Sunspot`	2K	Sunspot data
`Automobile`	32K	Automobile traffic sensor trace from large Midwestern state
`Temperature`	32K	Measurements from indoor temperature sensor (per second, deg. Celsius)

Table 12.3: Description of datasets (sizes are in number of points, 1K=1024 points).

Proof. On average, the wavelet crest scaling coefficients can be updated in $O(1)$ amortised time. Although a single step may require $O(\lg N)$ time in the worst case, on average, the (amortised) time required is $O(\sum_{i=0}^{n} \mathcal{B}(i)/N) = O(1)$ (where $\mathcal{B}(i)$ is the number of trailing zeros in the binary representation of i)[5]. Updating the $k \times k$ matrix for the appropriate linear equation (which can be identified in $O(1)$ time, based on the level l), requires time $O(k^2)$. □

Auto-regressive models with a comparable window size need $O(m^2k^2)$ time per update.

Corollary 1 (Constant-time update). *When the model parameters have been fixed (typically k is a small constant ≈ 6 and $m \sim \lg N$), the model requires space $O(\lg N)$ and amortised time $O(1)$ for each update.*

Figure 12.6(b) shows that this is clearly the case, as expected.

12.4 Experimental evaluation

We compared AWSOM against standard AR (with the equivalent, fair window size—see Section 12.3.4), as well as hand-tuned (S)ARIMA (wherever possible). Our prototype AWSOM implementation is written in Python, using Numeric Python for fast array manipulation. We used the standard `ts` package from R[6] for AR and (S)ARIMA models. We illustrate the properties of AWSOM and how to interpret the models using synthetic datasets and then show how these apply to real datasets (see Table 12.3).

Only the first half of each sequence was used to estimate the models, which were then applied to generate a sequence of length equal to that of the *entire* second half. For AR

[5]Seen differently, *IncDWT* is essentially a pre-order traversal of the wavelet coefficient tree.
[6]R version 1.6.0; see `http://www.r-project.org/`.

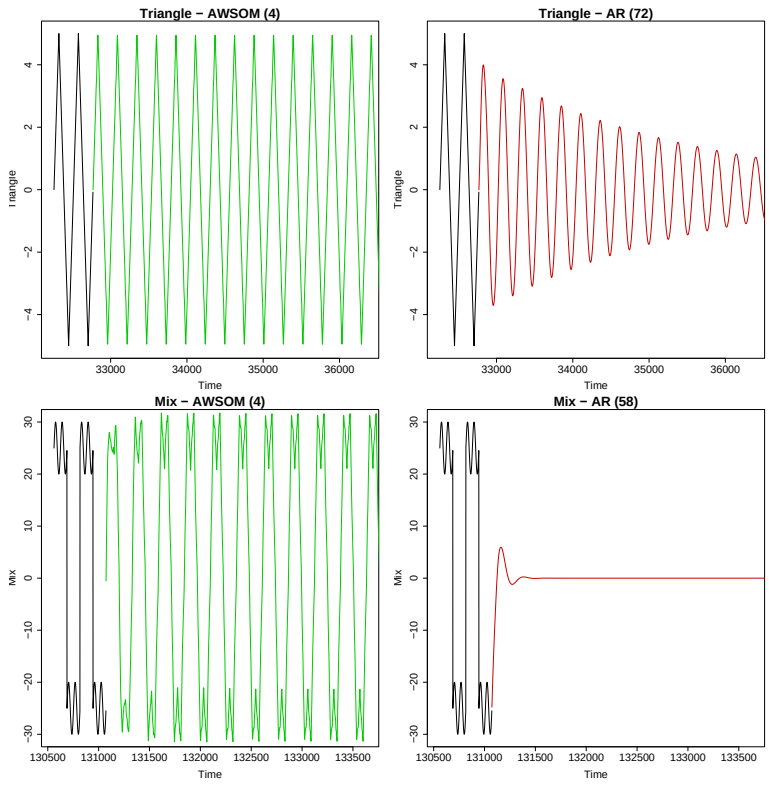
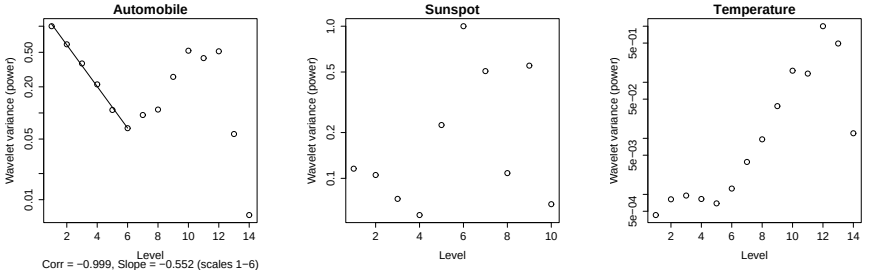


Figure 12.7: Forecasts—synthetic datasets. Note that AR gives the wrong trend (if any), while seasonal AR fails to complete and is not shown.

and (S)ARIMA, the last values (as dictated by the lags) of the first half were used to initiate generation. For AWSOM we again used as many of the last wavelet coefficients from each DWT level of the first half as were necessary to start applying the model equations. We should note that generating more than, say, 10 steps ahead is very rare: most methods in the literature [WG94] generate one step ahead, then obtain the correct value of X_{t+1}, and only then try to generate X_{t+2}. Nevertheless, our goal is to capture long-term behaviour under severe resource constraints and AWSOM achieves this efficiently.

12.4.1 Interpreting the models

Visual inspection A "forecast" is essentially a by-product of any *generative* time series model: application of any model to generate a number of "future" values reveals precisely the trends and patterns captured by that model. In other words, synthesising points based on the model is the simplest way for any user to get a quick, yet fairly accurate idea of what the trends are or, more precisely, what the *model* thinks they are. Thus, what we expect to



(a) `Automobile` (b) `Sunspot` (c) `Temperature`

Figure 12.8: Wavelet log-power diagnostic (real datasets). Horizontal axis is wavelet scale (l) and vertical axis is log-power ($\log \mathcal{V}_l$) (see Section 12.2.5). A linear trend with *negative* slope (between 0 and -1) indicates presence of self-similar noise. `Automobile` exhibits self-similarity in scales up to 6 (which roughly corresponds to one hour) but not overall.

see (especially in a long-range forecast) is the *important* patterns that can be identified from the real data. However, an expert user can extract even more precise information from the models.

Variance test As explained in Section 12.2.5, if the signal is self-similar, then the plot of log-power versus scale is linear.

Definition 27 (Variance log-power diagnostic). *The log-power vs. scale plot is the* wavelet variance log-power diagnostic plot *(or just* log-power diagnostic*). In particular, the correlation coefficient ρ_α quantifies the relation. If the plot is linear (in a range of scales), the slope $\hat{\alpha}$ is the* self-similarity exponent *($-1 < \alpha < 0$, closer to zero the more bursty the series).*

A large value of $|\rho_\alpha|$, at least across several scales, indicates that the series component in those scales may be modelled using, e.g., a fractional noise process with parameter dictated by α (see `Automobile`). However, we should otherwise be careful in drawing further conclusions about the behaviour within these scales.

We should note that after the observation by [LTWW94], fractional noise processes and, in general, self-similar sequences have revolutionised network traffic modelling. Furthermore, self-similar sequences appear in atomic clock fluctuations, river minima, compressed video bit-rates [Ber94, PW00], to mention a few examples.

Wavelet variance (energy and power) The magnitude of variance within each scale serves as an indicator about which frequency components are the dominant ones in the sequence. To precisely interpret the results, we also need to take into account the fundamental uncertainty in frequencies (see Figure 12.4). However, the wavelet variance plot quickly gives

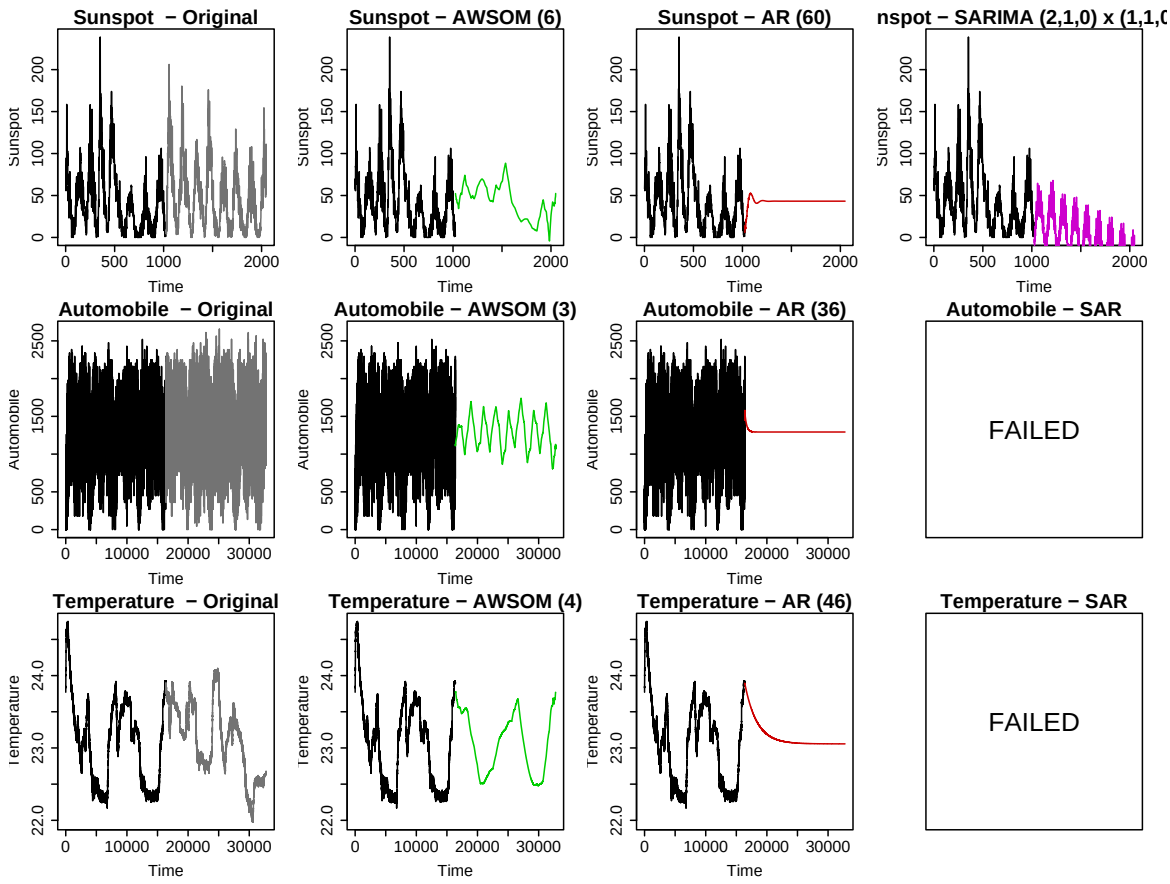


Figure 12.9: Forecasts—real datasets. AR fails to detect any trend, while seasonal AR fails to complete or gives a wrong conclusion in 260× time.

us the general picture of important trends. Furthermore, it guides us to focus on AWSOM coefficients around frequencies with large variance.

To summarise, the steps are: *(1)* Examine the log-power diagnostic to identify sub-bands that correspond to a self-similar component. These may be modelled using a fractional noise process for generation purposes; for forecasting purposes they are just that: noise. *(2)* Examine the wavelet energy spectrum to quickly identify important sub-bands.

Experimental goals. In each case, we demonstrate that AWSOM can provide information to answer the following questions, using *limited resources* and no supervision:

(Q1) Identify and capture periodic components: this can be done by simply inspecting the forecasts, or by examining the wavelet variance. The latter also gives information about the relative "significance" (essentially, amplitude) of each component.

(Q2) Diagnose the presence of self-similar noise in the appropriate scales: The log-power diagnostic provides the necessary information.

(Q3) Perform long-range forecasts: Besides identifying periodic components, the AWSOM coefficients at the appropriate scales capture their behaviour (*regardless* of their relative amplitude).

12.4.2 Synthetic datasets

We present synthetic datasets to illustrate the basic properties of AWSOM, its behaviour on several characteristic classes of sequences, and the principles behind interpreting the models. Applying the models to generate a number of "future" data points is the quickest way to see if each method captures long-term patterns.

`Triangle` AR fails to capture anything, because the window is not large enough. SAR estimation (with no differencing, no MA component and only a manually pre-specified 256-lag seasonal component) fails completely. In fact, R segfaults after several minutes, even without using maximum-likelihood estimation (MLE). However, AWSOM captures the periodicity.

This is immediately evident from the forecasts, which capture the trend almost perfectly. Also, inspection the wavelet variance (Figure 12.12(a)) shows a single spike at scale 7 (which corresponds to a window of $2^7 = 128$; this is as expected, since there is zero change among consecutive windows of the next size, 256).

`Mix` AR is again confused and does not capture even the sinusoidal component. SAR estimation (without MLE) fails (R's optimiser returns an error, after several minutes of computation).

Quick inspection of the AWSOM forecast shows clearly the two periodic components. The wavelet variance plot (Figure 12.12(b)) also gives this information: there is again a spike at scale 7 (or, window $2^7 = 128$; same interpretation as `Triangle`), which corresponds to the square wave periodic component. There is also a rise at scale 5 (or window $2^5 = 32$), which corresponds to the sinusoidal periodic component[7]). The difference in magnitude of the variances is precisely due to the difference in amplitude of the two periodic components: indeed, the square wave component is a much "stronger" one. However, regardless of the

[7]The non-zero value at scale 6 is expected and due to frequency leaks, as explained in Section 12.2.5.

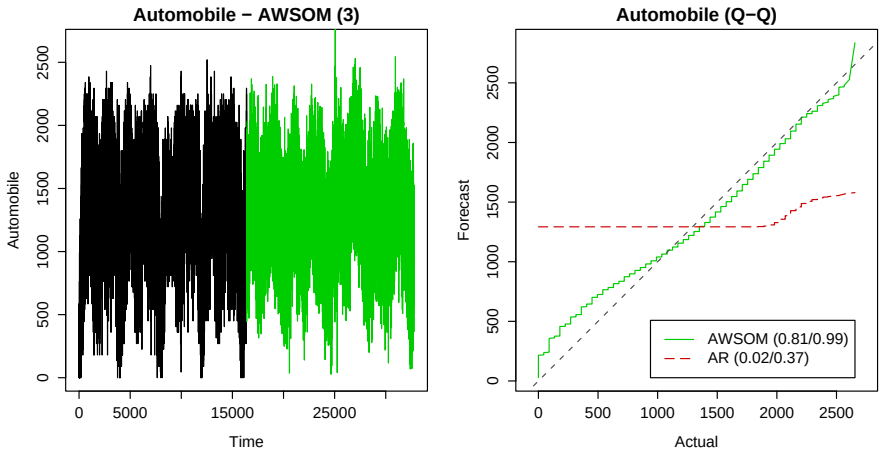


Figure 12.10: `Automobile`—generation with fractional noise.

strength (i.e., variance) of each component, the AWSOM coefficients capture their behaviour. In summary, using limited resources, AWSOM provides all the key information about the series.

12.4.3 Real datasets

For the real datasets, we show the marginal distribution *quantile-quantile plots* (or *Q-Q plots*—see Figure 12.11 and Figure 12.10)[8].

`Sunspot` This is a well-known dataset with a time-varying "period[9]." AR again fails completely. SAR (without a MA component, much less MLE) takes 40 minutes to estimate. AWSOM (in Python) takes less than 9 seconds. SAR misses the marginal distribution (see Figure 12.11) but, more importantly, it does not discover any period; that information has to be manually provided, after an initial inspection of the data. Furthermore, SAR can only deal successfully with one period only. AWSOM captures the general periodic trend, with a desirable slight confusion about the "period."

First, the log-power diagnostic (Figure 12.8(b)) shows some hint of self-similar noise at scales $2 - -4$ (and, indeed, the series has some fluctuation at the month scale). We won't focus on this behaviour here (see, however, discussion on `Automobile`) since the other

[8]These are the scatter plots of (x, y) such that p% of the values are below x in the real sequence and below y in the generated sequence. When the distributions are identical, the Q-Q plot coincides with the bisector of the first quadrant.

[9]Signals exhibiting this behaviour are often referred to as *cyclical*, as opposed to *periodic*

periodic (or, more accurately, cyclic) components are those that are of the most interest and dominate the series by far.

The wavelet variance (Figure 12.12(d)) indeed shows a spike in the vicinity of scale 6 (or window $2^6 = 64$). Each time tick is one month; it is a well-known fact (the so-called "Maunder minimum") that sunspots cycle at about 9–11.5 years (or, 108–138 months), with an average cycle length of about 10.8 years (approximately 129–130 months). Indeed, we see that the wavelet variance has a peak at windows of $64 - -128$ (scales $6 - -7$), which would correspond to repeating cycles every 128 or more months, as is the case.

Furthermore, the AWSOM coefficients at those scales capture the trend at that granularity (as can be seen immediately from the forecast), with the desirable "confusion" about the period.

Finally, closer examination of the series shows that the peaks at the last third are much lower than the rest of the series. This explains the other peak in Figure 12.12(d) at the scales of $9 - -10$, which correspond to a window of about 1/4–1/2 of the total series length.

`Automobile` This dataset has a strongly linear log-power diagnostic in scales 1–6 (Figure 12.8(a)). However, the lower frequencies (i.e., larger scales) contain the most energy (see Figure 12.12(e)). This indicates we should focus at these scales. The lowest frequency corresponds to a daily periodicity (approximately 4000 points per day, or about 8 periods in the entire series). The next highest frequency corresponds to the morning and afternoon rush-hours (approximately 2000 points per half-day). Also, we would expect to see a rise in the wavelet variance at scales corresponding to windows of $\approx 1000 - 2000$ or in the vicinity of scales $10 - -11$. Indeed, this is the case, and at these scales, the AWSOM coefficients capture the periodic components.

Furthermore, there appear to be significant differences in the dips between the two halves (this is clearer in Figure 12.1, where some of the noise has been removed), which explains the continued rise up to scale 12 (along with some small frequency leak, as explained in Section 12.2.5). This trend is not repeated frequently enough to be captured by a the regression models. However, the variance plot gives us a hint about this and the regression models on the other scales still do their job.

Next, we examine closer the self-similar noise at the hour scale (or high frequencies). In this series, these frequencies (corresponding scales 1–6) can be modelled by fractional noise. Figure 12.10 shows a generated sequence with fractional noise, as identified by AWSOM. The fractional difference parameter[10] is estimated as $\hat{\delta} \equiv -\hat{\alpha}/2 \approx 0.276$ and the amplitude

[10]This parameter is also related to the *Hurst exponent*.

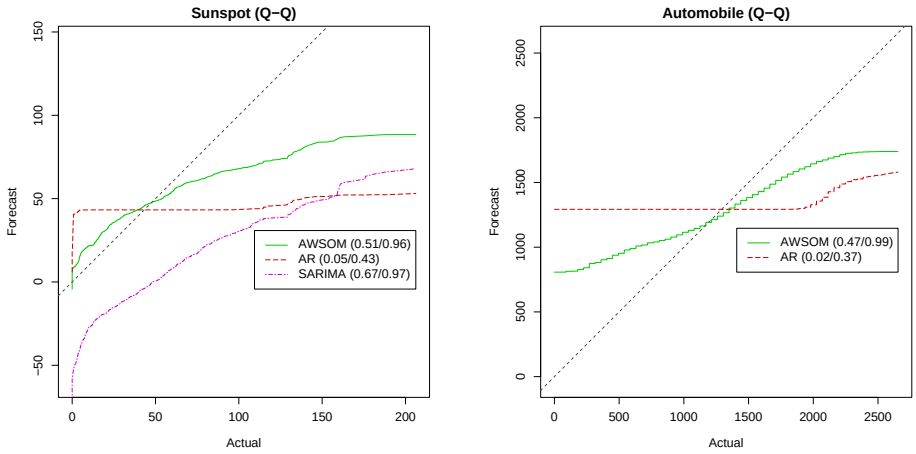


Figure 12.11: Marginal Q-Q plots (slope and correlation coefficients in parentheses).

is chosen to match the total variance in those scales.

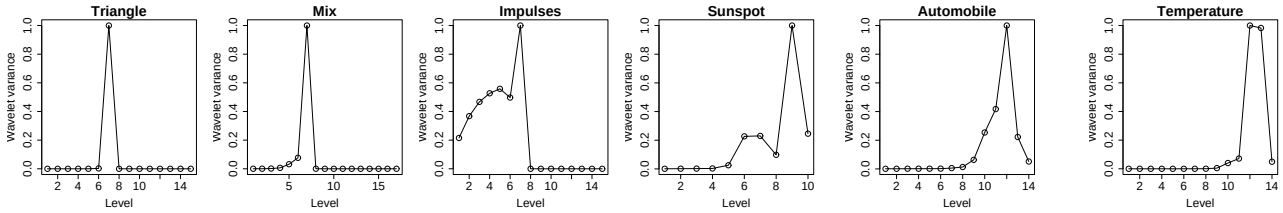
However, for unsupervised outlier detection, this is not necessary: what would really constitute an outlier would be, for instance, days that (a) do not follow the daily and rush-hour patterns, or (b) whose variance in the fractional noise scales is very different. This can be captured automatically by the series components in the appropriate frequency sub-bands that AWSOM identifies as a periodic component and bursty noise, respectively.

`Temperature` This data set consists of temperature measurements (in degrees Celsius) from small sensors that attach to the joystick port[11]. Each time tick is one second, thus the entire dataset is approximately 10 hours. The interpretation is similar to that of `Automobile`, except that there isn't a strong noise component at any scale (see Figure 12.8(c); the slope is *not* negative, as required for diagnosing self-similar noise). Also, observe that the wavelet variance (Figure 12.12(f)) tells us that the strongest trends happen at largest scales (from 10 and on, or windows > 1024). In other words, the most interesting activity is at times larger than about half an hour, which is indeed the case.

12.5 Discussion

In this section we first discuss correlations *across* scales (as opposed to *within* scales, which we have considered so far). Capturing linear correlations in this case shows some promise, provided that the signal is not extremely bursty (as can be partially "diagnosed" by the

[11] `http://www.ices.cmu.edu/sensornets/`



(a) `Triangle` (b) `Mix` (c) `Impulses` (d) `Sunspot` (e) `Automobile`(f) `Temperature`

Figure 12.12: Wavelet variances (average energy, normalised). The horizontal axis is wavelet level (l, i.e., wavelet window 2^l) and the vertical is wavelet variance (i.e., average of squares of wavelet coefficients for each l). The vertical axis is (linearly) normalised to a maximum of 1. Essentially, values at level l indicate "strength" of the series component at periods in the range $[2^l, 2^{l+1}]$ (see also Section 12.2.5).

Symbol	Definition
$\beta_{\delta l,\delta t}$	AWSOM coefficient, $(\delta l, \delta t) \in \mathcal{D}$).
$\mathcal{D}$	Set of window offsets for AWSOM.
AWSOM$(n_0,\ldots,n_\lambda)$	Offsets per level $(n_0,\ldots,n_\lambda)$ in $\mathcal{D}$—see Definition 28. $(n_0,\ldots,n_\lambda)$ is also called the AWSOM *order*.
λ	Depth of AWSOM model, $\lambda \geq 0$.
k	Total order of an AWSOM model $k \equiv \lvert\mathcal{D}\rvert = \sum_{l=0}^{\lambda} n_l$.

Table 12.4: Extra symbols and definitions (for models with inter-scale correlations).

variance criterion). More generally, we believe that further investigation of inter-scale correlations is a promising future direction.

Next, we discuss some evidence that illustrate why wavelet filters combined with linear regression per level actually have good modelling power, even under severe resource limitations.

12.5.1 Inter-scale correlations

In practice, correlations between scales are also occasionally present. These occur when there is a burst in the series; bursts localised in time have a presence in all frequencies (see also Figure 12.13(a)). If these bursts occur repeatedly and fairly consistently, then one possible approach for capturing them is to use linear regression not only within scales, but also across scales.

Formally, we try to fit models of the following form (see Table 12.4 for a summary of the main symbols):

$$W_{l,t} = \sum_{(\delta l,\delta t)\in\mathcal{D}} \beta^{(l)}_{\delta l,\delta t} W_{l+\delta l,t/2^{\delta l}-\delta t} + \epsilon_{l,t} \tag{12.3}$$

where $\mathcal{D}$ is a set of index offsets and $\epsilon_{l,t}$ is the usual error term. For example, in Figure 12.13(b), $\mathcal{D} = \{(0,1),(0,2),(1,0)\}$ and $W_{l,t} = \beta_{0,1}W_{l,t-1} + \beta_{0,2}W_{l,t-2} + \beta_{1,0}W_{l+1,t/2}$. The $\beta_{\delta l,\delta t}$ are called the AWSOM *coefficients*.

Definition 28 (AWSOM **order).** *The set of offsets is always of the form*

$$\begin{aligned}\mathcal{D} = \{ \quad & (0,1),\ (0,2),\ \ldots,\ (0,n_0),\\ & (1,0),\ (1,1),\ (1,2),\ \ldots,\ (1,n_1-1),\\ & \ldots,\\ & (\lambda,0),\ \ldots,\ (\lambda,n_\lambda-1)\ \}\end{aligned}$$

i.e., each wavelet coefficient is expressed as a function of the previous n_0 wavelet coefficients on the same level, n_1 coefficients from one level below and so on. For a particular choice of $\mathcal{D}$, we use

$$\text{AWSOM}(n_0, n_1, \ldots, n_\lambda)$$

to denote this instance of our model. We call $(n_0, \ldots, n_\lambda)$ the model's order. *The* total order *is the number of* AWSOM *coefficients k per equation, i.e., $k = \sum_{\delta l=0}^{\lambda} n_{\delta l}$ and λ is called the* depth *of the model.*

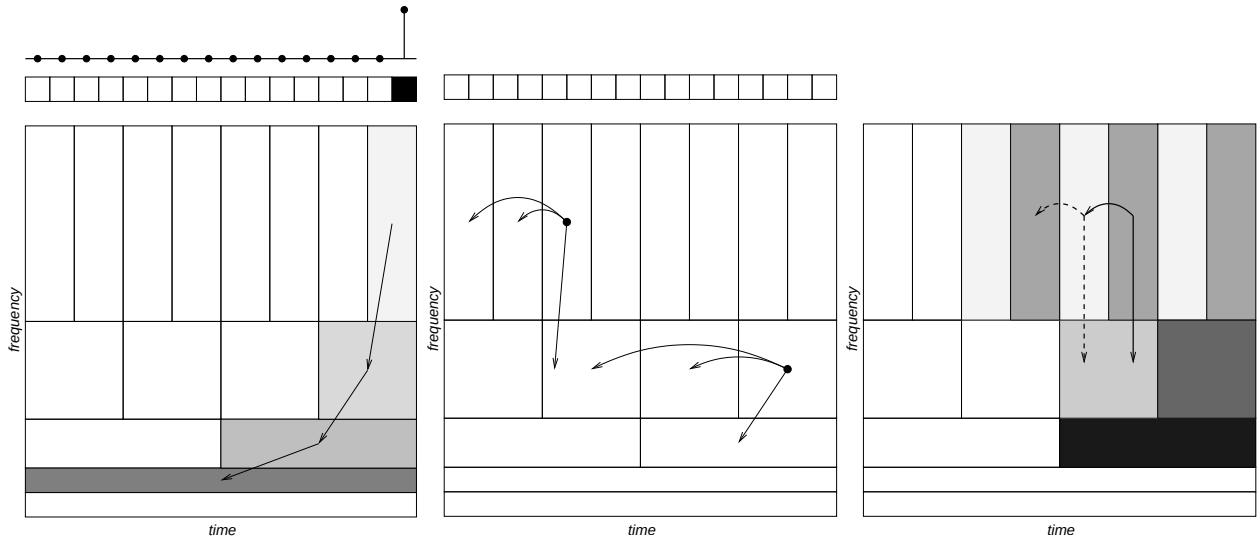
For example, Figure 12.13(b) shows an AWSOM$(2,1)$ model. A fixed choice of $\mathcal{D}$ is sufficient for all signals. In practice, we use one inter-scale coefficient, corresponding to offset $(1,0)$.

Furthermore, we fit one equation per level (see Figure 12.13(b)), as long as the level contains enough wavelet coefficients to get a good fit[12]. Thus, we fit one equation for every level $l < L_a$. These are the *active levels*, where L_a is the level that has no more than, say, $N_a = 16$ wavelet coefficients. For levels $l \geq L_a$ (the *inactive levels*), we can either keep the exact wavelet coefficients (which would be no more than $16 + 8 + \cdots + 1 = 31$ in the above case) and/or fit one more equation.

In other words, the number of inactive levels L_i is always fixed to, say, $L_i = 4$ and the number of active levels L_a grows as the stream size increases.

When fitting an AWSOM model with depth $\lambda \geq 1$, we also fit different equations depending on time location t. For instance, if we are using AWSOM$^1(n_0, 2)$, we should fit one equation for pairs $W_{l,2t}$ and $W_{l-1,t}$ and another for pairs $W_{l,2t+1}$ and $W_{l-1,t}$ (see Figure 12.13(c)).

[12]However, the remaining coefficients may still provide important information about the signal. Furthermore, they will be later used to fit models, as more points arrive.



(a) (b) (c)

Figure 12.13: (a) Inter-scale correlations, intuition. (b,c) Illustration of AWSOM$(1,1)$ with $L_i = 2$ inactive levels. The shade of each wavelet coefficient corresponds to the model equation used to "predict" it. The unshaded wavelet coefficients correspond to initial conditions (i.e., with incomplete AWSOM window $\mathcal{D}$).

In general, we need 2^λ separate models to ensure that the inter-scale correlations λ levels down are not "shoehorned" into the same regression model.

To summarise, the inter-scale AWSOM model fits a number of equations:

$$W_{l,t} = \sum_{(\delta l,\delta t)\in\mathcal{D}} \beta_{\delta l,\delta t}^{(l',t')} W_{l+\delta l,t-\delta t} \epsilon_{l,t} \tag{12.4}$$

for $l' \leq L_a$ and $t' \equiv t \bmod T$, $0 \leq t' < T$. For example, if $T = 2$, we estimate one linear equation for each set of wavelet coefficients $W_{0,2i}$, $W_{0,2i+1}$, $W_{l,2i}$ and $W_{l,2k+1}$ ($l \geq 1$, $i \geq 0$). The significant advantage of this approach is that we can still easily update the AWSOM equations online, as new data values arrive. This is possible because the equation is selected based only on l and t for the new wavelet coefficient.

Complexity The complexity with respect to stream size does not change. In particular, the number of model equations now is $m = L_a T$ where

$$T \sim 2^\lambda \quad \text{and} \quad L_a \sim \lg \frac{N}{N_a T} = \lg N - \lg N_\Lambda - \lambda.$$

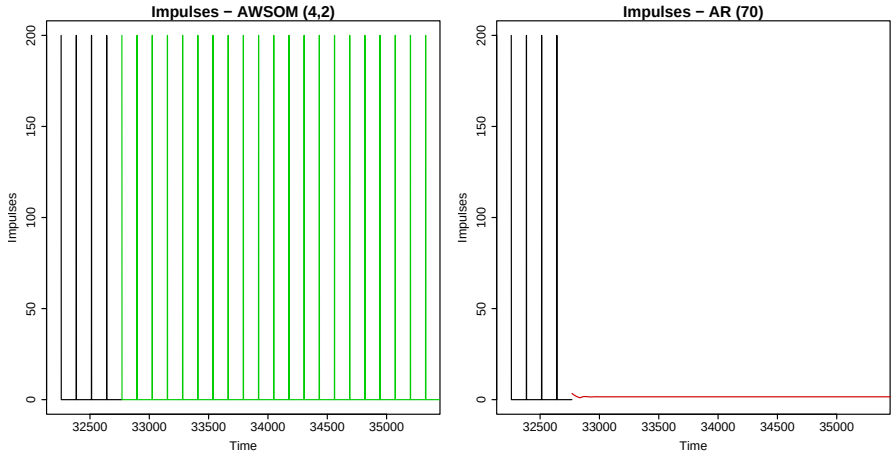


Figure 12.14: Forecasts—`Impulses`. The signal consists of an impulse train (every 256 points), for a total of 64K points.

Once again, $m = O(\lg N)$, since λ is fixed and depends only on the AWSOM order. The number of AWSOM coefficients k is now the total order of the model. Thus, once again, the space required with respect to stream is $O(\lg N)$. The linear equation to be updated can, again, be identified in $O(1)$ time (using $t \bmod T$, as well as the level l), so the update with respect to stream size still requires only $O(1)$ amortised time.

Model combination Model selection as presented in Section 12.3.3 can be extended to include model combination. The only thing that needs to be decided in advance is the largest AWSOM$(n_0, \ldots, n_\lambda)$ order we may want to fit. From the data collected, we can then automatically select any model of smaller order (AWSOM$(n'_0, \ldots, n'_{\lambda'})$, where $\lambda' \leq \lambda$ and $n'_i \leq n_i$).

If we split measurements x_i into two subsets $\mathbf{X}_1$ and $\mathbf{X}_2$ with corresponding outputs $\mathbf{y}_1$ and $\mathbf{y}_2$, then the LS solution for both subsets combined is given by

$$\mathbf{b} = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{y},$$

where

$$\mathbf{X} = \begin{bmatrix}\mathbf{X}_1^T & \mathbf{X}_2^T\end{bmatrix}^T \text{ and } \mathbf{y} = [\mathbf{y}_1^T \ \mathbf{y}_2^T]^T$$

i.e.,

$$\begin{aligned}\mathbf{b} &= (\mathbf{X}_1^T\mathbf{X}_1 + \mathbf{X}_2^T\mathbf{X}_2)^{-1}(\mathbf{X}_1^T\mathbf{y}_1 + \mathbf{X}_2^T\mathbf{y}_2) \\ &= (\mathbf{P}_1 + \mathbf{P}_2)^{-1}(\mathbf{q}_1 + \mathbf{q}_2).\end{aligned}$$

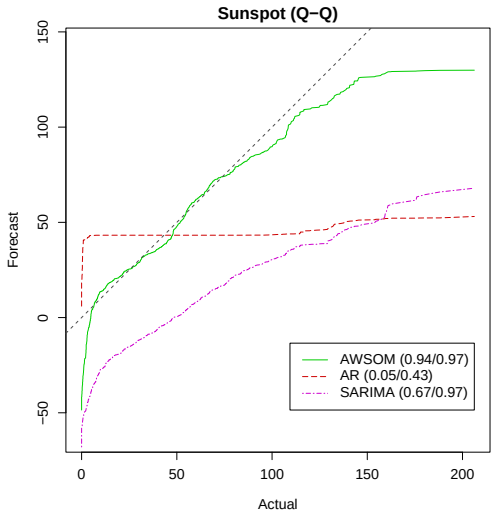


Figure 12.15: Marginal Q-Q plots for `Sunspot` with inter-scale correlations.

Therefore, it is possible to combine sub-models when reducing the number of levels (effectively reducing $T \equiv 2^{\lambda}$).

Experimental evaluation

We show inter-scale AWSOM models for `Sunspot` and `Temperatures` using one inter-scale coefficient (i.e., instead of AWSOM(k), we use AWSOM$(k, 1)$—in all cases, a single inter-scale coefficient is sufficient). Also, for illustrative purposes, we show one synthetic dataset, `Impulses` (based on Figure 12.13(a)).

In summary, inter-scale coefficients, used in conjunction with the log-power diagnostic, can be helpful in better capturing the nature of the signal. The use of inter-scale coefficients does not change anything in the previous discussions of questions (Q1)–(Q3). The only further consideration here is the presence of repeated bursts and strong dips or rises.

`Impulses` This synthetic dataset (see Figure 12.14) consists of single impulses that repeat every 256 time ticks. It illustrates how inter-scale correlations may help (see Figure 12.13(a)). AR fails to capture anything (again, too small window) and SAR estimation fails, while AWSOM captures the overall behaviour.

Regarding the wavelet variance, we indeed see a peak at scale 7 (i.e., window $2^7 = 128$; again, wavelet consecutive windows of 256 at the next scale show no change, so the wavelet coefficients are zero, as expected). The other non-zero values at lower scales are also expected. As noted before, periodic components are generally well-isolated in frequency (or,

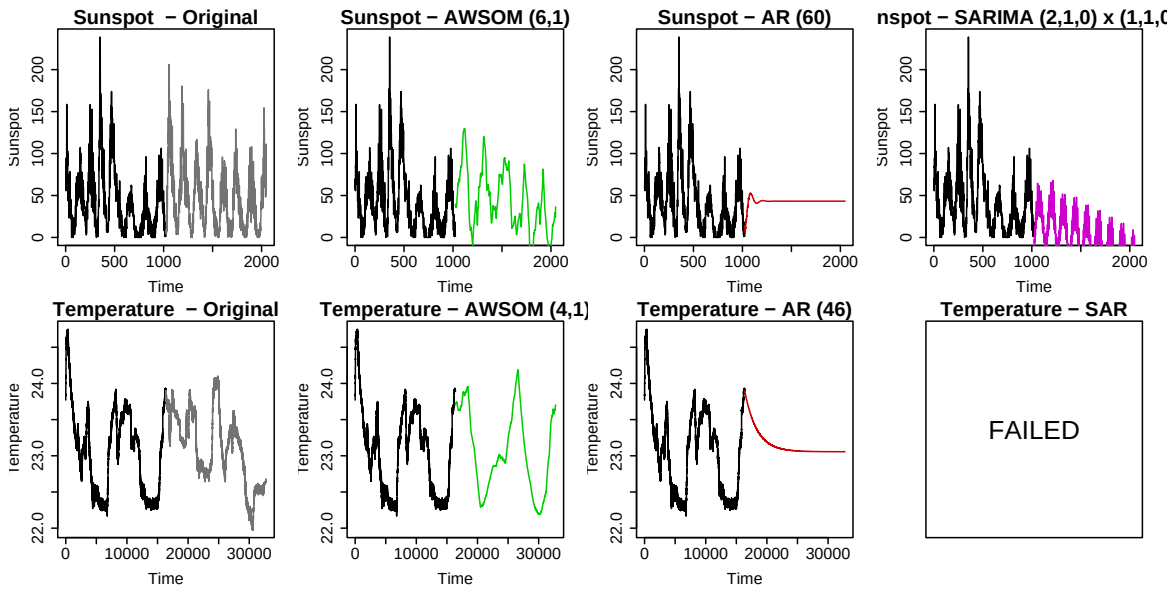


Figure 12.16: Forecasts—`Sunspot` and `Temperature` with inter-scale correlations.

scale), but a *single* impulse in time has presence in *all* frequencies; *repeated* impulses show presence only at frequencies *higher* than the impulse frequency. So, this is normal and, in fact, it is *precisely* why inter-scale correlations are helpful in these cases.

`Sunspot` and `Temperature` In both cases, the inter-scale coefficient captures some of the burstiness present in the signal. This is illustrated more clearly in `Sunspot`. Note that the detection of periodic trends is not affected. However, the dips and spikes of the signal are captured at varying degrees.

12.5.2 Why AWSOM makes sense

In this section, we discuss in more detail the properties of the DWT and the intuition behind AWSOM, using real signals to demonstrate the key points. The *auto-correlation function (ACF)* and *partial ACF (PACF)* are explained briefly in Section 12.2.2. In summary, the PACF for a pure auto-regressive AR(p) process is non-zero for lags up to p. For a pure moving average (MA) process, the PACF decays exponentially. Conversely, the ACF for a pure auto-regressive (AR) process decays exponentially. For a pure moving average process MA(q), the ACF is non-zero for lags up to q. For a pure white noise process, the ACF is zero for all lags. The dashed lines in the graphs indicate confidence intervals.

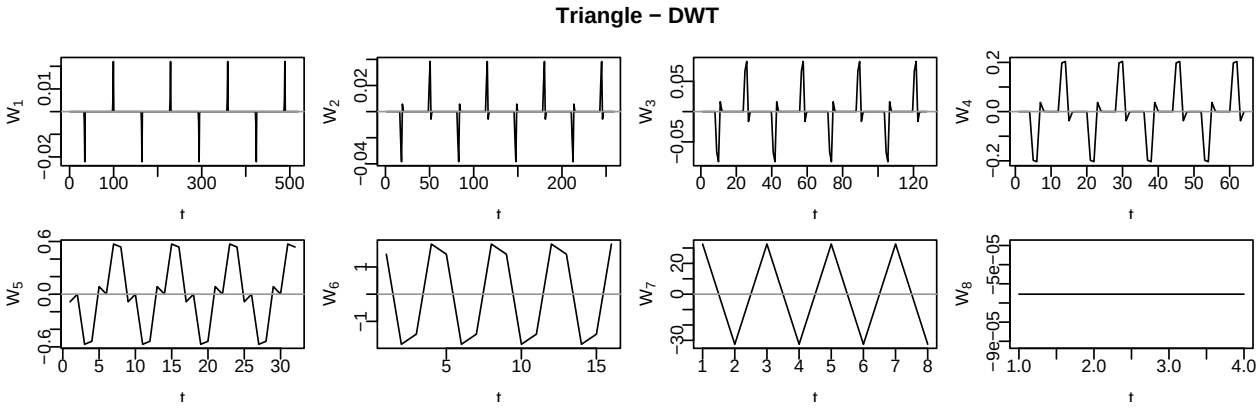


Figure 12.17: DWT of `Triangle` (note that each level is plotted at a different scale; coefficients on levels other than 7 are negligible).

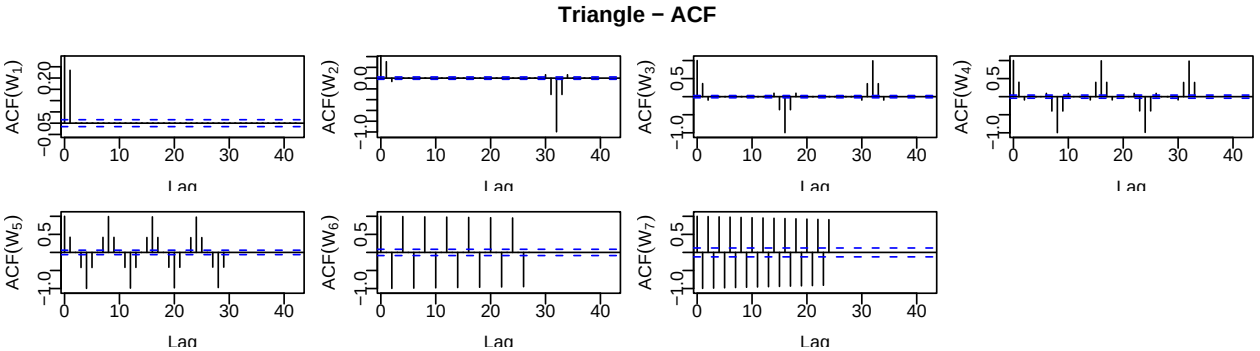


Figure 12.18: ACF (per DWT level) for `Triangle`.

`Triangle` Figure 12.17 shows the DWT coefficients (in particular, a large enough fragment so that the plots and patterns are clearly distinguishable). The main behaviour of the signal is reflected in level 7 (which corresponds to frequencies $[1/64, 1/128]$ or a window of 128) and that an AWSOM(1) model would be sufficient to capture it. In particular, $W_{7,t} = -W_{7,t-1}$ and, in fact, with feature selection this is precisely the equation we get. The PACF (see Figure 12.19) also indicates that low-order auto-regressive models are sufficient for several levels. However, the non-zero values on all levels except 7 are almost negligible and that is generally the case for signals that are fairly smooth at an appropriate scale[13]. Therefore, failing to capture them perfectly has a small impact.

[13]For regularly repeating bursts, see the preceding discussion in Section 12.5.1.

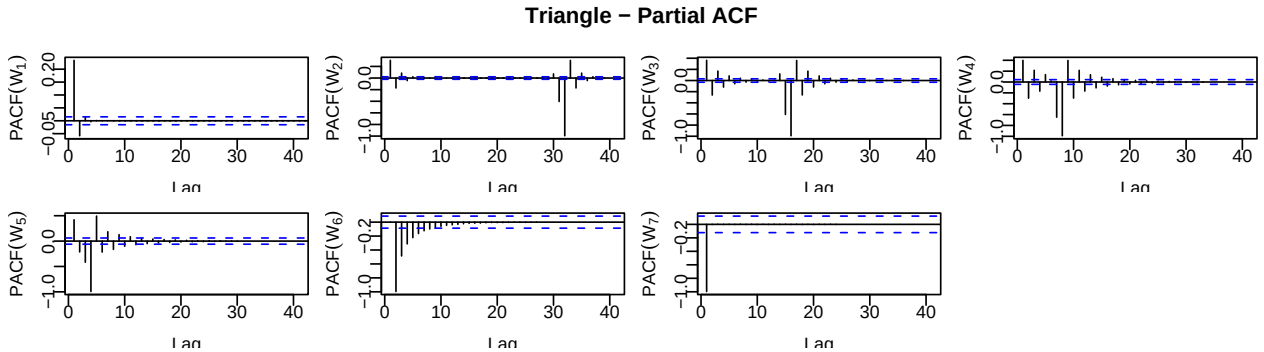


Figure 12.19: Partial ACF (per DWT level) for `Triangle`.

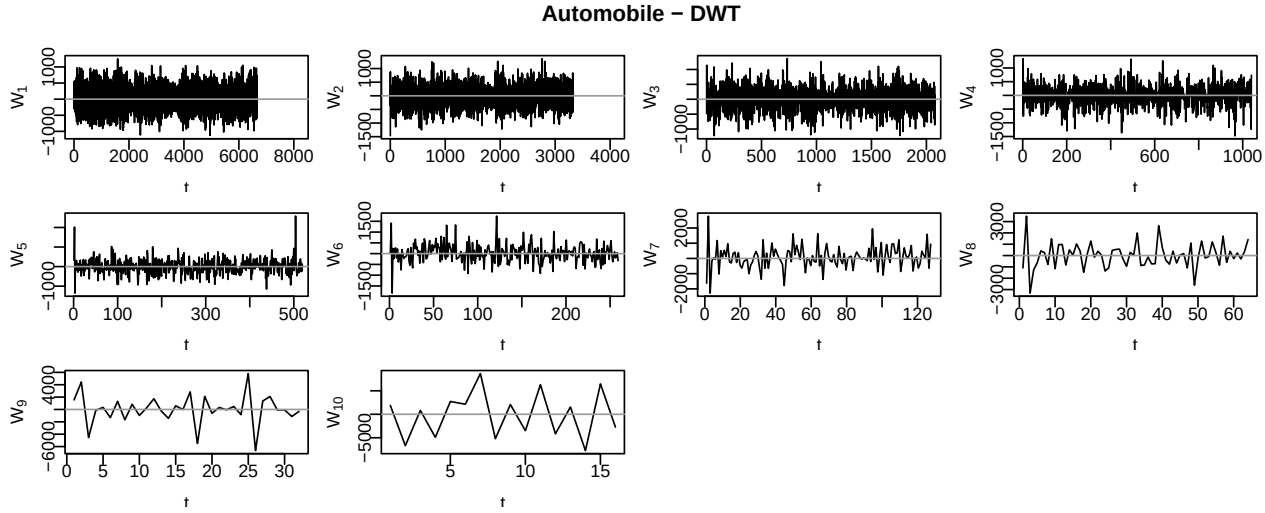


Figure 12.20: DWT of `Automobile`.

`Automobile` Figure 12.20 shows the complete DWT. Among these, the largest variance belongs to levels 10 and 9. The PACF (Figure 12.22) provides some indication that an AWSOM(3) model would be sufficient to capture most of the behaviour at level 10 (also, note that the ACF decays). This can also be seen to some extent in the DWT plot (in fact, the largest AWSOM coefficient at that level corresponds to lag 3). At level 9, all three coefficients are kept by feature selection, but the largest one corresponds to lag 1, as expected.

General remarks First, the DWT does not decorrelate the signals; if that was the case, then we should have $\rho_k = 0$ for all $k \geq 1$. However, the DWT does successfully filter out the noise and capture the main behaviour of the signal at the appropriate levels (which generally correspond to the largest variance). This is demonstrated by the plots of the DWT

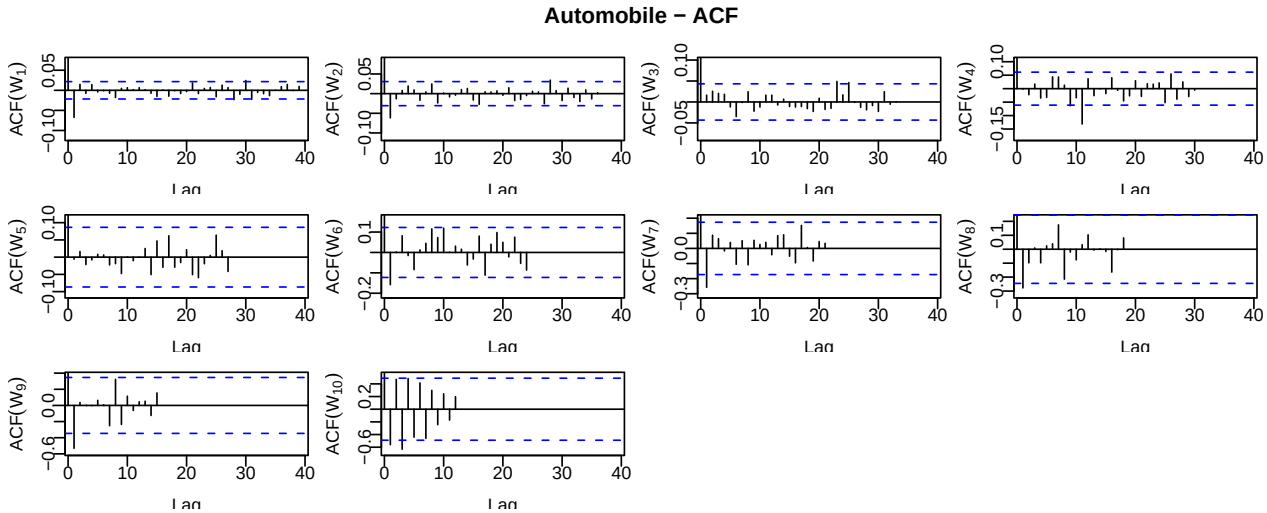


Figure 12.21: ACF (per DWT level) for `Automobile`.

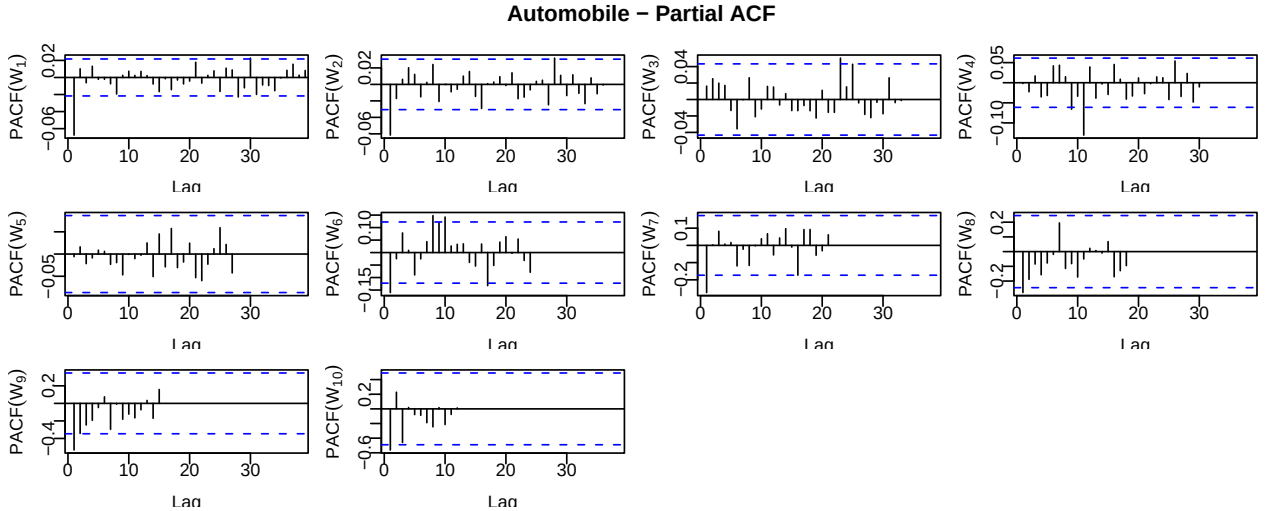


Figure 12.22: Partial ACF (per DWT level) for `Automobile`.

levels. Furthermore, the ACF also indicates presence of long-range dependencies. Finally, it shows that AR models typically suffice to capture the behaviour of the key components of the signal (i.e., the levels with the largest wavelet coefficients, which correspond to the largest variance).

Finally, the information we keep for an AWSOM(p) model is sufficient to compute the ACF up to lag p, as well as the PACF up to the same lag (see also Section 12.3.3—the **P** matrix keeps all the necessary auto-covariance information). This can be used after the log-power diagnostic to extract further information.

12.6 Conclusions

Sensor networks are becoming increasingly popular, thanks to falling prices and increasing storage and processing power. We presented AWSOM, which achieves all of the following goals:

1. *Concise patterns:* AWSOM provides linear models with few coefficients, it can detect arbitrary periodic components, it gives information across several frequencies and it can diagnose self-similarity and long-range dependence.

2. *Streaming framework:* We can update patterns in an "any-time" fashion, with one pass over the data, in time independent of stream size and using $O(\lg N)$ space (where N is the length of the sequence so far). Furthermore, AWSOM can do forecasting (directly, for the estimated model).

3. *Unsupervised operation:* Once we decide the largest AWSOM order, no further intervention is needed; the sensor can be left alone to collect information.

We showed real and synthetic data, where our method captures the periodicities and burstiness, while manually selected AR (or even (S)ARIMA generalisations, which are not suitable for streams with limited resources) fails completely.

AWSOM is an important first step toward hands-off stream mining, combining simplicity with modelling power. Continuous queries are useful for evidence gathering and hypothesis testing *once* we know what we are looking for. AWSOM is the first method to deal directly with the problem of unsupervised stream mining and pattern detection and fill the gap.

Chapter 13

Correlations among multiple streams

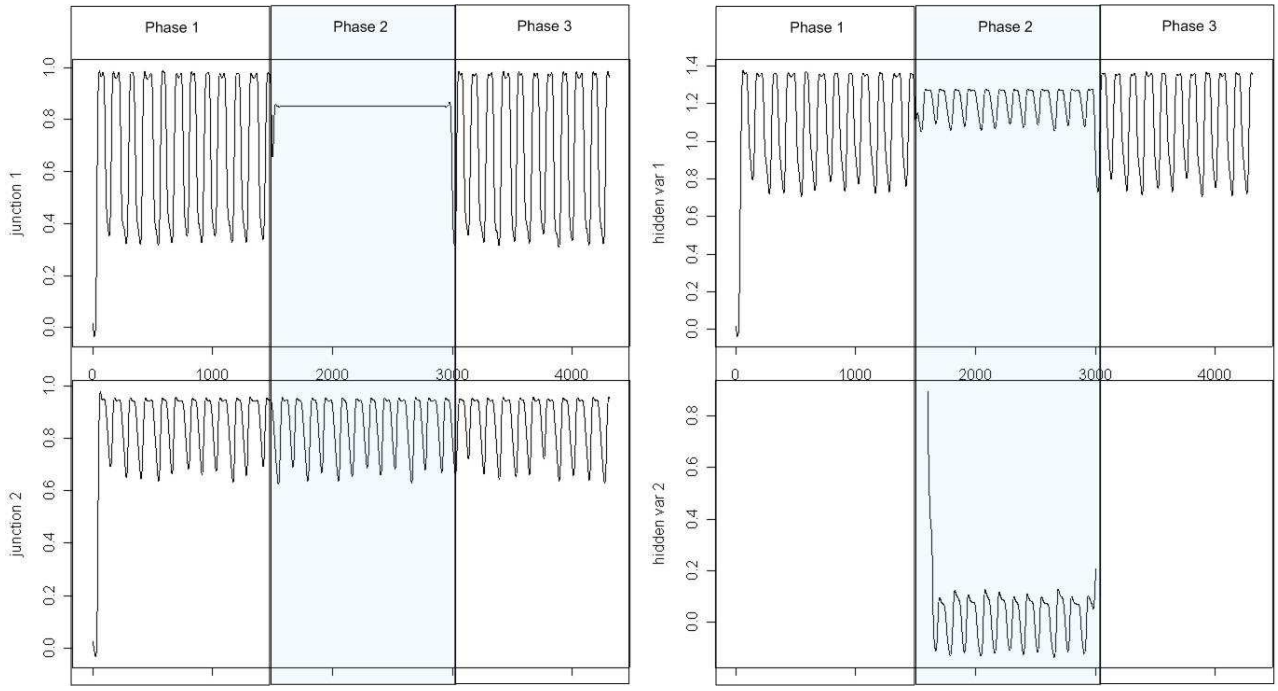
In this chapter, we consider the problem of capturing correlations and finding hidden variables corresponding to trends on collections of semi-infinite, time series data streams, where the data consist of tuples with n numbers, one for each time tick t.

Streams often are inherently correlated (e.g., temperatures in the same building, traffic in the same network, prices in the same market, etc.) and it is possible to reduce hundreds of numerical streams into just a handful of *hidden variables* that compactly describe the key trends and dramatically reduce the complexity of further data processing. We propose an approach to do this incrementally.

We describe a motivating scenario, to illustrate the problem we want to solve. Consider a large number of sensors measuring chlorine concentration in a drinkable water distribution network (see Figure 13.1, showing 15 days worth of data). Every five minutes, each sensor sends its measurement to a central node, which monitors and analyses the streams in real time.

The patterns in chlorine concentration levels normally arise from water demand. If water is not refreshed in the pipes, existing chlorine reacts with pipe walls and micro-organisms and its concentration drops. However, if fresh water flows in at a particular location due to demand, chlorine concentration rises again. The rise depends primarily on how much chlorine is originally mixed at the reservoirs (and also, to a small extent, on the distance to the closest reservoir—as the distance increases, the peak concentration drops slightly, due to reactions along the way). Thus, since demand typically follows a periodic pattern, chlorine concentration reflects that (see Figure 13.1(a), bottom): it is high when demand is high and vice versa.

Assume that at some point in time, there is a major leak at some pipe in the network. Since fresh water flows in constantly (possibly mixed with debris from the leak), chlorine



(a) Sensor measurements (b) Hidden variables

Figure 13.1: Illustration of problem. Sensors measure chlorine in drinking water and show a daily, near sinusoidal periodicity during phases 1 and 3. During phase 2, some of the sensors are "stuck" due to a major leak. The extra hidden variable introduced during phase 2 captures the presence of a new trend. SPIRIT can also tell us which sensors participate in the new, "abnormal" trend (e.g., close to a construction site). In phase 3, everything returns to normal.

concentration at the nodes near the leak will be close to peak at all times.

Figure 13.1(a) shows measurements collected from two nodes, one away from the leak (bottom) and one close to the leak (top). At any time, a human operator would like to know how many trends (or *hidden variables*) are in the data and ask queries about them. Each hidden variable essentially corresponds to a group of correlated streams.

In this simple example, SPIRIT discovers the correct number of hidden variables. Under normal operation, only one hidden variable is needed, which corresponds to the periodic pattern (Figure 13.1(b), top). Both observed variables follow this hidden variable (multiplied by a constant factor, which is the *participation weight* of each observed variable into the particular hidden variable). Mathematically, the hidden variables are the *principal components* of the observed variables and the participation weights are the entries of the *principal direction* vectors[1].

[1]More precisely, this is true under certain assumptions, which will be explained later.

However, during the leak, a second trend is detected and a new hidden variable is introduced (Figure 13.1(b), bottom). As soon as the leak is fixed, the number of hidden variables returns to one. If we examine the hidden variables, the interpretation is straightforward: The first one still reflects the periodic demand pattern in the sections of the network under normal operation. All nodes in this section of the network have a participation weight of ≈ 1 to the "periodic trend" hidden variable and ≈ 0 to the new one. The second hidden variable represents the additive effect of the catastrophic event, which is to cancel out the normal pattern. The nodes close to the leak have participation weights ≈ 0.5 to both hidden variables.

Summarising, SPIRIT can tell us that (Figure 13.1):

- Under normal operation (phases 1 and 3), there is one trend. The corresponding hidden variable follows a periodic pattern and all nodes participate in this trend. All is well.

- During the leak (phase 2), there is a *second* trend, trying to cancel the normal trend. The nodes with non-zero participation to the corresponding hidden variable can be immediately identified (e.g., they are close to a construction site). An abnormal event may have occurred in the vicinity of those nodes, which should be investigated.

Matters are further complicated when there are hundreds or thousands of nodes and more than one demand pattern. However, as we show later, SPIRIT is still able to extract the key trends from the stream collection, follow trend drifts and immediately detect outliers and abnormal events.

Besides providing a concise summary of key trends/correlations among streams, SPIRIT can successfully deal with missing values and its discovered hidden variables can be used to do very efficient, resource-economic forecasting.

Of course, there are several other applications and domains to which SPIRIT can be applied. For example, (i) given more than 50,000 securities trading in US, on a second-by-second basis, detect patterns and correlations [ZS02], (ii) given traffic measurements [YSJ+00], find routers that tend to go down together.

Contributions

The problem of pattern discovery in a large number of co-evolving streams has attracted much attention in many domains. We introduce *SPIRIT (Streaming Pattern dIscoveRy in mul-*

tIple Time-series), a comprehensive approach to discover correlations that effectively and efficiently summarise large collections of streams. SPIRIT satisfies the following requirements:

- It is *streaming*, i.e., it is incremental, scalable, *any-time*. It requires very memory and processing time per time tick. In fact, both are independent of the stream length t.
- It scales *linearly* with the number of streams n, not quadratically. This may seem counter-intuitive, because the naïve method to spot correlations across n streams examines all $O(n^2)$ pairs.
- It is *adaptive*, and fully *automatic*. It dynamically detects changes (both gradual, as well as sudden) in the input streams, and automatically determines the number k of hidden variables.

The correlations and hidden variables we discover have multiple uses. They provide a succinct summary to the user, they can help to do fast forecasting and detect outliers, and they facilitate interpolations and handling of missing values, as we discuss later.

The rest of the chapter is organised as follows: Section 13.1 discusses related work, on data streams and stream mining. Section 13.2 overviews some of the background and explains the intuition behind our approach. Section 13.3 describes our method and Section 13.4 shows how its output can be interpreted and immediately utilised, both by humans, as well as for further data analysis. Section 13.5 discusses experimental case studies that demonstrate the effectiveness of our approach. In Section 13.6 we elaborate on the efficiency and accuracy of SPIRIT. Finally, in Section 13.7 we conclude.

13.1 Related work

There is a large body of work on streams, which we loosely classify in two groups.

Data stream management systems (DSMS) We include this very broad category for completeness. DSMS include Aurora [ACC+03], Stream [MWA+03], Telegraph [CCD+03] and Gigascope [CJSS03]. The common hypothesis is that (i) massive data streams come into the system at a very fast rate, and (ii) near real-time monitoring and analysis of incoming data streams is required. The new challenges have made researchers re-think many parts of traditional DBMS design in the streaming context, especially on query processing using correlated attributes [DGMH05], scheduling [BO03, CCR+03], load shedding [TCZ+03, DGR03] and memory requirements [ABB+02].

In addition to system-building efforts, a number of approximation techniques have been studied in the context of streams, such as sampling [BDM02], sketches [DGGR02, CDIM02, GGR03], exponential histograms [DGI+02], and wavelets [GKS04]. The main goal of these methods is to estimate a global aggregate (e.g. sum, count, average) over a window of size w on the recent data. The methods usually have resource requirements that are sublinear with respect to w. Most focus on a single stream.

The emphasis in this line of work is to support traditional SQL queries on streams. None of them try to find patterns, nor to do forecasting.

Data mining on streams Researchers have started to redesign traditional data mining algorithms for data streams. Much of the work has focused on finding interesting patterns in a single stream, but multiple streams have also attracted significant interest. Ganti et al. [GGR02] propose a generic framework for stream mining. Guha et al. [GMM+03] propose a one-pass k-median clustering algorithm. Domingos and Hulten [DH00] construct a decision tree online, by passing over the data only once. Recently, [HSD01, WFYH03] address the problem of finding patterns over concept drifting streams. In Chapter 12 we proposed a method to find patterns in a single stream, using wavelets [PBF03]. More recently, Palpanas et al. [PVK+04] consider approximation of time-series with *amnesic* functions. They propose novel techniques suitable for streaming, and applicable to a wide range of user-specified approximating functions.

Keogh et al. [KLR04b] propose parameter-free methods for classic data mining tasks (i.e., clustering, anomaly detection, classification), based on compression. Lin et al. [LVKG04] perform clustering on different levels of wavelet coefficients of multiple time series. Both approaches require having all the data in advance. Recently, Ali et al. [AMAK05] propose a framework for *Phenomena Detection and Tracking (PDT)* in sensor networks. They define a phenomenon on discrete-valued streams and develop query execution techniques based on multi-way hash join with PDT-specific optimisations.

CluStream [AHY03] is a flexible clustering framework with online and offline components. The online component extends micro-cluster information [ZRL96] by incorporating exponentially-sized sliding windows while coalescing micro-cluster summaries. Actual clusters are found by the offline component. StatStream [ZS02] uses the DFT to summarise streams within a finite window and then compute the highest pairwise correlations among all pairs of streams, at each timestamp. Very recently, BRAID [SPF05b] addresses the problem of discovering lag correlations among multiple streams. The focus is on time and space efficient methods for finding the earliest and highest peak in the cross-correlation functions

between all pairs of streams. Neither CluStream, StatStream or BRAID explicitly focus on discovering hidden variables.

Guha et al. [GGK03] improve on discovering correlations, by first doing dimensionality reduction with random projections, and then periodically computing the SVD. However, the method incurs high overhead because of the SVD re-computation and it can not easily handle missing values. MUSCLES [YSJ+00] is exactly designed to do forecasting (thus it could handle missing values). However, it can not find hidden variables and it scales poorly for a large number of streams n, since it requires at least quadratic space and time, or expensive reorganisation (*selective MUSCLES*).

Finally, a number of the above methods usually require choosing a sliding window size, which typically translates to buffer space requirements. Our approach does not require any sliding windows and does not need to buffer *any* of the stream data.

In conclusion, none of the above methods simultaneously satisfy the requirements in the introduction: "any-time" streaming operation, scalability on the number of streams, adaptivity, and full automation.

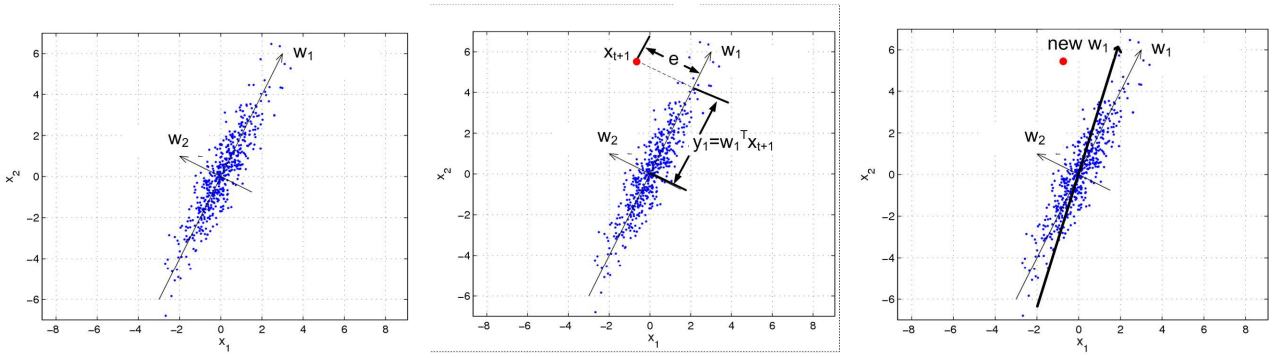
13.2 Principal component analysis (PCA)

Here we give a brief overview of PCA [Jol02] and explain the intuition behind our approach. We use standard matrix algebra notation: vectors are lower-case bold, matrices are upper-case bold, and scalars are in plain font. The transpose of matrix $\mathbf{X}$ is denoted by $\mathbf{X}^T$. In the following, $\mathbf{x}_t \equiv [x_{t,1}\, x_{t,2}\, \cdots\, x_{t,n}]^T \in \mathbb{R}^n$ is the column-vector[2] of stream values at time t. The stream data can be viewed as a continuously growing $t \times n$ matrix $\mathbf{X}_t := [\mathbf{x}_1\, \mathbf{x}_2\, \cdots\, \mathbf{x}_t]^T \in \mathbb{R}^{t \times n}$, where one new row is added at each time tick t. In the chlorine example, $\mathbf{x}_t$ is the measurements column-vector at t over all the sensors, where n is the number of chlorine sensors and t is the measurement timestamp.

Typically, in collections of n-dimensional points $\mathbf{x}_t \equiv [x_{t,1} \ldots, x_{t,n}]^T$, $t = 1, 2, \ldots$, there exist correlations between the n dimensions (which correspond to streams in our setting). These can be captured by principal components analysis (PCA). Consider for example the setting in Figure 13.2. There is a visible linear correlation. Thus, if we represent every point with its projection on the direction of $\mathbf{w}_1$, the error of this approximation is very small. In fact, the first principal direction $\mathbf{w}_1$, is the *optimal* in the following sense.

Definition 29 (First principal component). *Given a collection of n-dimensional vectors* $\mathbf{x}_\tau \in \mathbb{R}^n$,

[2]We adhere to the common convention of using column vectors and writing them out in transposed form.



(a) Original $\mathbf{w}_1$ (b) Update process (c) Resulting $\mathbf{w}_1$

Figure 13.2: Illustration of updating $\mathbf{w}_1$ when a new point $\mathbf{x}_{t+1}$ arrives.

$\tau = 1, 2, \ldots, t$, *the* first principal direction $\mathbf{w}_1 \in \mathbb{R}^n$ *is the vector that minimises the sum of squared residuals, i.e.,*

$$\mathbf{w}_1 := \arg\min_{\|w\|=1} \sum_{\tau=1}^{t} \|\mathbf{x}_\tau - (\mathbf{w}\mathbf{w}^T)\mathbf{x}_\tau\|^2.$$

The projection of $\mathbf{x}_\tau$ *on* $\mathbf{w}_1$ *is the* first principal component (PC) $y_{\tau,1} := \mathbf{w}_1^T \mathbf{x}_\tau$, $\tau = 1, \ldots, t$.

Note that, since $\|\mathbf{w}_1\| = 1$, we have $(\mathbf{w}_1\mathbf{w}_1^T)\mathbf{x}_\tau = (\mathbf{w}_1^T\mathbf{x}_\tau)\mathbf{w}_1 = y_{\tau,1}\mathbf{w}_1 =: \tilde{\mathbf{x}}_\tau$, where $\tilde{\mathbf{x}}_\tau$ is the projection of $y_{\tau,1}$ back into the original n-D space. That is, $\tilde{\mathbf{x}}_\tau$ is the *reconstruction* of the original measurements from the first PC $y_{\tau,1}$. More generally, PCA will produce k vectors $\mathbf{w}_1, \mathbf{w}_2, \ldots, \mathbf{w}_k$ such that, if we represent each n-D data point $\mathbf{x}_t := [x_{t,1} \cdots x_{t,n}]$ with its k-D projection $\mathbf{y}_t := [\mathbf{w}_1^T\mathbf{x}_t \cdots \mathbf{w}_k^T\mathbf{x}_t]^T$, then this representation minimises the squared error $\sum_\tau \|\mathbf{x}_t - \tilde{\mathbf{x}}_t\|^2$. Furthermore, the principal directions are orthogonal, so the principal components $y_{\tau,i}, 1 \le i \le k$ are *by construction uncorrelated*, i.e., if $\mathbf{y}^{(i)} := [y_{1,i}, \ldots, y_{t,i}, \ldots]^T$ is the stream of the i-th principal component, then $\left(\mathbf{y}^{(i)}\right)^T \mathbf{y}^{(j)} = 0$ if $i \neq j$.

Observation 2 (Dimensionality reduction). *If we represent each n-dimensional point* $\mathbf{x}_\tau \in \mathbb{R}^n$ *using all n principal components, then the error* $\|\mathbf{x}_\tau - \tilde{\mathbf{x}}_\tau\| = 0$. *However, in typical datasets, we can achieve a very small error using only k principal components, where* $k \ll n$.

In the context of the chlorine example, each point in Figure 13.2 would correspond to the 2-D projection of $\mathbf{x}_\tau$ (where $1 \le \tau \le t$) onto the first two principal directions, $\mathbf{w}_1$ and $\mathbf{w}_2$, which are the most important according to the distribution of $\{\mathbf{x}_\tau \mid 1 \le \tau \le t\}$. The principal components $y_{\tau,1}$ and $y_{\tau,2}$ are the coordinates of these projections in the orthogonal coordinate system defined by $\mathbf{w}_1$ and $\mathbf{w}_2$.

However, batch methods for estimating the principal components require time that depends on the duration t, which grows to infinity. In fact, the principal directions are the

Symbol	Description
$\mathbf{x}, \ldots$	Column vectors (lowercase boldface).
$\mathbf{A}, \ldots$	Matrices (uppercase boldface).
$\mathbf{x}_t$	The n stream values $\mathbf{x}_t := [x_{t,1} \cdots x_{t,n}]^T$ at time t.
n	Number of streams.
$\mathbf{w}_i$	The i-th participation weight vector (i.e., principal direction).
k	Number of hidden variables.
$\mathbf{y}_t$	$\mathbf{y}_t \equiv [y_{t,1} \cdots y_{t,k}]^T := [\mathbf{w}_1^T \mathbf{x}_t \cdots \mathbf{w}_k^T \mathbf{x}_t]^T$.
$\tilde{\mathbf{x}}_t$	Reconstruction of $\mathbf{x}_t$ from the k hidden variable values, i.e., $\tilde{\mathbf{x}}_t := y_{t,1}\mathbf{w}_1 + \cdots + y_{t,k}\mathbf{w}_k$.
E_t	Total energy up to time t.
$\tilde{E}_{t,i}$	Total energy captured by the i-th hidden variable, up to time t.
f_E, F_E	Lower and upper bounds on the fraction of energy we wish to maintain via SPIRIT's approximation.

Table 13.1: Description of notation.

eigenvectors of $\mathbf{X}_t^T \mathbf{X}_t$, which are best computed through the singular value decomposition (SVD) of $\mathbf{X}_t$. Space requirements also depend on t. Clearly, in a stream setting, it is impossible to perform this computation at every step, aside from the fact that we don't have the space to store all past values. In short, we want a method that does not need to store *any* past values.

13.3 Tracking correlations and hidden variables: SPIRIT

In this section we present our framework for discovering patterns in multiple streams. In the next section, we show how these can be used to perform effective, low-cost forecasting. We use auto-regression for its simplicity, but our framework allows any forecasting algorithm to take advantage of the compact representation of the stream collection.

Problem definition Given a collection of n co-evolving, semi-infinite streams, producing a value $x_{t,j}$, for every stream $1 \leq j \leq n$ and for every time-tick $t = 1, 2, \ldots$, SPIRIT does the following:

- Adapts the number k of *hidden variables* necessary to explain/summarise the main trends in the collection.

- Adapts the *participation weights* $w_{i,j}$ of the j-th stream on the i-th hidden variable ($1 \leq j \leq n$ and $1 \leq i \leq k$), so as to produce an accurate summary of the stream collection.
- Monitors the hidden variables $y_{t,i}$, for $1 \leq i \leq k$.
- Keeps updating all the above efficiently.

More precisely, SPIRIT operates on the column-vectors of observed stream values $\mathbf{x}_t \equiv [x_{t,1}, \ldots, x_{t,n}]^T$ and continually updates the participation weights $w_{i,j}$. The *participation weight vector* $\mathbf{w}_i$ for the i-th principal direction is $\mathbf{w}_i := [w_{i,1} \cdots w_{i,n}]^T$. The hidden variables $\mathbf{y}_t \equiv [y_{t,1}, \ldots, y_{t,k}]^T$ are the projections of $\mathbf{x}_t$ onto each $\mathbf{w}_i$, over time (see Table 13.1), i.e.,

$$y_{t,i} := w_{i,1}x_{t,1} + w_{i,2}x_{t,2} + \cdots + w_{i,n}x_{t,n},$$

SPIRIT also adapts the number k of hidden variables necessary to capture most of the information. The adaptation is performed so that the approximation achieves a desired mean-square error. In particular, let $\tilde{\mathbf{x}}_t = [\tilde{x}_{t,1} \cdots \tilde{x}_{t,n}]^T$ be the *reconstruction* of $\mathbf{x}_t$, based on the weights and hidden variables, defined by

$$\tilde{x}_{t,j} := w_{1,j}y_{t,1} + w_{2,j}y_{t,2} + \cdots + w_{k,j}y_{t,k},$$

or more succinctly, $\tilde{\mathbf{x}}_t = \sum_{i=1}^{k} y_{i,t}\mathbf{w}_i$.

In the chlorine example, $\mathbf{x}_t$ is the n-dimensional column-vector of the original sensor measurements and $\mathbf{y}_t$ is the hidden variable column-vector, both at time t. The dimension of $\mathbf{y}_t$ is 1 before/after the leak ($t < 1500$ or $t > 3000$) and 2 during the leak ($1500 \leq t \leq 3000$), as shown in Figure 13.1.

Definition 30 (SPIRIT tracking). *SPIRIT updates the participation weights $w_{i,j}$ so as to guarantee that the reconstruction error $\|\tilde{\mathbf{x}}_t - \mathbf{x}_t\|^2$* over time *is predictably small.*

This informal definition describes what SPIRIT does. The precise criteria regarding the reconstruction error will be explained later. If we assume that the $\mathbf{x}_t$ are drawn according to some distribution that does not change over time (i.e., under *stationarity* assumptions), then the weight vectors $\mathbf{w}_i$ converge to the principal directions. However, even if there are non-stationarities in the data (i.e., gradual drift), in practice we can deal with these very effectively, as we explain later.

An additional complication is that we often have missing values, for several reasons: either failure of the system, or delayed arrival of some measurements. For example, the

sensor network may get overloaded and fail to report some of the chlorine measurements in time or some sensor may temporarily black-out. At the very least, we want to continue processing the rest of the measurements.

13.3.1 Tracking the hidden variables

The first step is, for a given k, to incrementally update the k participation weight vectors $\mathbf{w}_i$, $1 \leq i \leq k$, so as to summarise the original streams with only a few numbers (the hidden variables). In Section 13.3.2, we describe the complete method, which also adapts k.

For the moment, assume that the number of hidden variables k is given. Furthermore, our goal is to minimise the average reconstruction error $\sum_t \|\tilde{\mathbf{x}}_t - \mathbf{x}_t\|^2$. In this case, the desired weight vectors $\mathbf{w}_i, 1 \leq i \leq k$ are the principal directions and it turns out that we can estimate them incrementally.

We use an algorithm based on adaptive filtering techniques [Yan95, Hay92], which have been tried and tested in practice, performing well in a variety of settings and applications (e.g., image compression and signal tracking for antenna arrays). We experimented with several alternatives [Oja89, DK96] and found this particular method to have the best properties for our setting: it is very efficient in terms of computational and memory requirements, while converging quickly, with no special parameters to tune. The main idea behind the algorithm is to read in the new values $\mathbf{x}_{t+1} \equiv [x_{(t+1),1}, \ldots, x_{(t+1),n}]^T$ from the n streams at time $t+1$, and perform three steps:

1. Compute the hidden variables $y'_{t+1,i}, 1 \leq i \leq k$, based on the *current* weights $\mathbf{w}_i, 1 \leq i \leq k$, by projecting $\mathbf{x}_{t+1}$ onto these.

2. Estimate the reconstruction error ($\mathbf{e}_i$ below) and the energy, based on the $y'_{t+1,i}$ values.

3. Update the estimates of $\mathbf{w}_i, 1 \leq i \leq k$ and output the *actual* hidden variables $y_{t+1,i}$ for time $t+1$.

To illustrate this, Figure 13.2(b) shows the $\mathbf{e}_1$ and y_1 when the new data $\mathbf{x}_{t+1}$ enter the system. Intuitively, the goal is to adaptively update $\mathbf{w}_i$ so that it quickly converges to the "truth." In particular, we want to update $\mathbf{w}_i$ more when $\mathbf{e}_i$ is large. However, the magnitude of the update should also take into account the past data currently "captured" by $\mathbf{w}_i$. For this reason, the update is inversely proportional to the current *energy* $E_{t,i}$ of the i-th hidden variable, which is $E_{t,i} := \frac{1}{t}\sum_{\tau=1}^{t} y_{\tau,i}^2$. Figure 13.2(c) shows $\mathbf{w}_1$ after the update for $\mathbf{x}_{t+1}$.

Algorithm TRACKW

0. Initialise the k hidden variables $\mathbf{w}_i$ to unit vectors $\mathbf{w}_1 = [10\cdots0]^T$, $\mathbf{w}_2 = [010\cdots0]^T$, etc. Initialise d_i ($i = 1, ...k$) to a small positive value. Then:
1. As each point $\mathbf{x}_{t+1}$ arrives, initialise $\acute{\mathbf{x}}_1 := \mathbf{x}_{t+1}$.
2. For $1 \leq i \leq k$, we perform the following assignments and updates, in order:

$$\begin{aligned}
y_i &:= \mathbf{w}_i^T \acute{\mathbf{x}}_i && (y_{t+1,i} = \text{projection onto } \mathbf{w}_i)\\
d_i &\leftarrow \lambda d_i + y_i^2 && (\text{energy} \propto i\text{-th eigenval. of } \mathbf{X}_t^T\mathbf{X}_t)\\
\mathbf{e}_i &:= \acute{\mathbf{x}}_i - y_i\mathbf{w}_i && (\text{error, } \mathbf{e}_i \perp \mathbf{w}_i)\\
\mathbf{w}_i &\leftarrow \mathbf{w}_i + \frac{1}{d_i} y_i \mathbf{e}_i && (\text{update PC estimate})\\
\acute{\mathbf{x}}_{i+1} &:= \acute{\mathbf{x}}_i - y_i\mathbf{w}_i && (\text{repeat with remainder of } \mathbf{x}_t).
\end{aligned}$$

The *forgetting factor* λ will be discussed in Section 13.3.3 (for now, assume $\lambda = 1$). For each i, $d_i = tE_{t,i}$ and $\acute{\mathbf{x}}_i$ is the component of $\mathbf{x}_{t+1}$ in the orthogonal complement of the space spanned by the updated estimates $\mathbf{w}_{i'}, 1 \leq i' < i$ of the participation weights. The vectors $\mathbf{w}_i, 1 \leq i \leq k$ are in order of importance (more precisely, in order of decreasing eigenvalue or energy). It can be shown that, under stationarity assumptions, these $\mathbf{w}_i$ in these equations converge to the true principal directions.

Complexity We only need to keep the k weight vectors $\mathbf{w}_i$ ($1 \leq i \leq k$), each n-dimensional. Thus the total cost is $O(nk)$, both in terms of time and of space. The update cost does not depend on t. This is a tremendous gain, compared to the usual PCA computation cost of $O(tn^2)$.

13.3.2 Detecting the number of hidden variables

In practice, we do not know the number k of hidden variables. We propose to estimate k on the fly, so that we maintain a high percentage f_E of the *energy* E_t. Energy thresholding is a common method to determine how many principal components are needed [Jol02]. Formally, the energy E_t (at time t) of the sequence of $\mathbf{x}_t$ is defined as

$$E_t := \tfrac{1}{t}\sum_{\tau=1}^{t} \|\mathbf{x}_\tau\|^2 = \tfrac{1}{t}\sum_{\tau=1}^{t}\sum_{i=1}^{n} x_{\tau,i}^2.$$

Similarly, the energy $\tilde{E}_t$ of the reconstruction $\tilde{\mathbf{x}}$ is defined as

$$\tilde{E}_t := \tfrac{1}{t}\sum_{\tau=1}^{t} \|\tilde{\mathbf{x}}_\tau\|^2.$$

Lemma 13. *Assuming the* $\mathbf{w}_i, 1 \leq i \leq k$ *are orthonormal, we have*

$$\tilde{E}_t = \frac{1}{t}\sum_{\tau=1}^{t} \|\mathbf{y}_\tau\|^2 = \frac{t-1}{t}\tilde{E}_{t-1} + \frac{1}{t}\|\mathbf{y}_t\|.$$

Proof. If the $\mathbf{w}_i, 1 \leq i \leq k$ are orthonormal, then it follows easily that $\|\tilde{\mathbf{x}}_\tau\|^2 = \|y_{\tau,1}\mathbf{w}_1 + \cdots + y_{\tau,k}\mathbf{w}_k\|^2 = y_{\tau,1}^2\|\mathbf{w}_1\|^2 + \cdots + y_{\tau,k}^2\|\mathbf{w}_k\|^2 = y_{\tau,1}^2 + \cdots + y_{\tau,k}^2 = \|\mathbf{y}_\tau\|^2$ (Pythagorean theorem and normality). The result follows by summing over τ. □

It can be shown that algorithm TRACKW maintains orthonormality without the need for any extra steps (otherwise, a simple re-orthonormalisation step at the end would suffice).

From the user's perspective, we have a low-energy and a high-energy threshold, f_E and F_E, respectively. We keep enough hidden variables k, so the retained energy is within the range $[f_E \cdot E_t, F_E \cdot E_t]$. Whenever we get outside these bounds, we increase or decrease k. In more detail, the steps are:

1. Estimate the full energy E_{t+1}, incrementally, from the sum of squares of $x_{\tau,i}$.
2. Estimate the energy $\tilde{E}_{(k)}$ of the k hidden variables.
3. Possibly, adjust k. We introduce a new hidden variable (update $k \leftarrow k+1$) if the current hidden variables maintain too little energy, i.e., $\tilde{E}_{(k)} < f_E E$. We drop a hidden variable (update $k \leftarrow k-1$), if the maintained energy is too high, i.e., $\tilde{E}_{(k)} > F_E E$.

The energy thresholds f_E and F_E are chosen according to recommendations in the literature [Jol02, Fuk90]. We use a lower energy threshold $f_E = 0.95$ and an upper energy threshold $F_E = 0.98$. Thus, the reconstruction $\tilde{\mathbf{x}}_t$ retains between 95% and 98% of the energy of $\mathbf{x}_t$.

Algorithm SPIRIT

0. Initialise $k \leftarrow 1$ and the total energy estimates of $\mathbf{x}_t$ and $\tilde{\mathbf{x}}_t$ per time tick to $E \leftarrow 0$ and $\tilde{E}_1 \leftarrow 0$. Then,

1. As each new point arrives, update $\mathbf{w}_i$, for $1 \leq i \leq k$ (step 1, TRACKW).

2. Update the estimates (for $1 \leq i \leq k$)

$$E \leftarrow \frac{(t-1)E + \|\mathbf{x}_t\|^2}{t} \quad \text{and} \quad \tilde{E}_i \leftarrow \frac{(t-1)\tilde{E}_i + y_{t,i}^2}{t}.$$

3. Let the estimate of retained energy be

$$\tilde{E}_{(k)} := \sum_{i=1}^{k} \tilde{E}_i.$$

If $\tilde{E}_{(k)} < f_E E$, then we start estimating $\mathbf{w}_{k+1}$ (initialising as in step 0 of TRACKW), initialise $\tilde{E}_{k+1} \leftarrow 0$ and increase $k \leftarrow k+1$. If $\tilde{E}_{(k)} > F_E E$, then we discard $\mathbf{w}_k$ and $\tilde{E}_k$ and decrease $k \leftarrow k-1$.

The following lemma proves that the above algorithm guarantees the relative reconstruction error is within the specified interval $[f_E, F_E]$.

Lemma 14. *The relative squared error of the reconstruction satisfies*

$$1 - F_E \leq \frac{\sum_{\tau=1}^{t} \|\tilde{\mathbf{x}}_\tau - \mathbf{x}_\tau\|^2}{\sum_t \|\mathbf{x}_\tau\|^2} \leq 1 - f_E.$$

Proof. From the orthogonality of $\mathbf{x}_\tau$ and the complement $\tilde{\mathbf{x}}_\tau - \mathbf{x}_\tau$ we have $\|\tilde{\mathbf{x}}_\tau - \mathbf{x}_\tau\|^2 = \|\mathbf{x}_\tau\|^2 - \|\tilde{\mathbf{x}}_\tau\|^2 = \|\mathbf{x}_\tau\|^2 - \|\mathbf{y}_\tau\|^2$ (by Lemma 13). The result follows by summing over τ and from the definitions of E and $\tilde{E}$. □

Finally, in Section 13.6.2 we demonstrate that the incremental weight estimates are extremely close to the principal directions computed with offline PCA.

13.3.3 Exponential forgetting

We can adapt to more recent behaviour by using an exponential forgetting factor $0 < \lambda < 1$. This allows us to follow trend drifts over time. We use the same λ for the estimation of both $\mathbf{w}_i$ as well as the AR models (see Section 13.4.1). However, we also have to properly keep track of the energy, discounting it with the same rate, i.e., the update at each step is:

$$E \leftarrow \frac{\lambda(t-1)E + \|\mathbf{x}_t\|^2}{t} \quad \text{and} \quad \tilde{E}_i \leftarrow \frac{\lambda(t-1)\tilde{E}_i + y_{t,i}^2}{t}.$$

Typical choices are $0.96 \leq \lambda \leq 0.98$ [Hay92]. As long as the values of $\mathbf{x}_t$ do not vary wildly, the exact value of λ is not crucial. We use $\lambda = 0.96$ throughout. A value of $\lambda = 1$ makes sense when we know that the sequence is stationary (rarely true in practice, as most sequences gradually drift). Note that the value of λ does not affect the computation cost of our method. In this sense, an exponential forgetting factor is more appealing than a sliding window, as the latter has explicit buffering requirements.

13.4 Putting SPIRIT to work

We show how we can exploit the correlations and hidden variables discovered by SPIRIT to do (a) forecasting, (b) missing value estimation, (c) summarisation of the large number of streams into a small, manageable number of hidden variables, and (d) outlier detection.

13.4.1 Forecasting and missing values

The hidden variables $\mathbf{y}_t$ give us a much more compact representation of the "raw" variables $\mathbf{x}_t$, with guarantees of high reconstruction accuracy (in terms of relative squared error, which is less than $1 - f_E$). When our streams exhibit correlations, as we often expect to be the case, the number k of the hidden variables is much smaller than the number n of streams. Therefore, we can apply *any* forecasting algorithm to the vector of hidden variables $\mathbf{y}_t$, instead of the raw data vector $\mathbf{x}_t$. This reduces the time and space complexity by orders of magnitude, because typical forecasting methods are quadratic or worse on the number of variables.

In particular, we fit the forecasting model on the $\mathbf{y}_t$ instead of $\mathbf{x}_t$. The model provides an estimate $\hat{\mathbf{y}}_{t+1} = f(\mathbf{y}_t)$ and we can use this to get an estimate for

$$\hat{\mathbf{x}}_{t+1} := \hat{y}_{t+1,1}\mathbf{w}_1[t] + \cdots + \hat{y}_{t+1,1}\mathbf{w}_k[t],$$

using the weight estimates $\mathbf{w}_i[t]$ from the previous time tick t. We chose auto-regression for its intuitiveness and simplicity, but any online method can be used.

Correlations Since the principal directions are orthogonal ($\mathbf{w}_i \perp \mathbf{w}_j, i \neq j$), the components of $\mathbf{y}_t$ are *by construction uncorrelated*—the correlations have already been captured by the $\mathbf{w}_i, 1 \leq i \leq k$. We can take advantage of this de-correlation reduce forecasting complexity. In particular for auto-regression, we found that one AR model per hidden variable provides results comparable to multivariate AR.

Auto-regression Space complexity for multivariate AR (e.g., MUSCLES [YSJ+00]) is $O(n^3\ell^2)$, where ℓ is the auto-regression window length. For AR per stream (ignoring correlations), it is $O(n\ell^2)$. However, for SPIRIT, we need $O(kn)$ space for the $\mathbf{w}_i$ and, with one AR model per y_i, the total space complexity is $O(kn + k\ell^2)$. As published, MUSCLES requires space that grows cubically with respect to the number of streams n. We believe it can be made to work with quadratic space, but this is still prohibitive. Both AR per stream and SPIRIT require space that grows linearly with respect to n, but in SPIRIT k is typically very small

($k \ll n$) and, in practice, SPIRIT requires less memory and time per update than AR per stream. More importantly, a single, independent AR model per stream cannot capture *any* correlations, whereas SPIRIT indirectly exploits the correlations present *within* a time tick.

Missing values When we have a forecasting model, we can use the forecast based on $\mathbf{x}_{t-1}$ to estimate missing values in $\mathbf{x}_t$. We then use these estimated missing values to update the weight estimates, as well as the forecasting models. Forecast-based estimation of missing values is the most time-efficient choice and gives very good results.

13.4.2 Interpretation

At *any* given time t, SPIRIT readily provides two key pieces of information (aside from the forecasts, etc.):

- The number of hidden variables k.
- The weights $w_{i,j}$, $1 \leq i \leq k$, $1 \leq j \leq n$. Intuitively, the magnitude $|w_{i,j}|$ of each weight tells us how much the i-th hidden variable contributes to the reconstruction of the j-th stream.

In the chlorine example during phase 1 (see Figure 13.1), the dataset has only one hidden variable, because one sinusoidal-like pattern can reconstruct both streams (albeit with different weights for each). Thus, SPIRIT correctly identifies correlated streams. When the correlation was broken, SPIRIT introduces enough hidden variables to capture that. Finally, it also spots that, in phase 3, normal operation is reestablished and thus disposes of the unnecessary hidden variable. In Section 13.5 we show additional examples of how we can intuitively interpret this information.

13.5 Experimental case studies

In this section we present case studies on real and realistic datasets to demonstrate the effectiveness of our approach in discovering the underlying correlations among streams. In particular, we show that:

- We capture the appropriate number of hidden variables. As the streams evolve, we capture these changes in real-time [SPF05c] and adapt the number of hidden variables k and the weights $\mathbf{w}_i$.

Dataset	n	k	Description
`Chlorine`	166	2	Chlorine concentrations from EPANET.
`Critter`	8	1–2	Temperature sensor measurements.
`River`	3	1	River gauge data from USACE.
`Motes`	54	2–4	Light sensor measurements.

Table 13.2: Description of datasets.

- We capture the essential behaviour with very few hidden variables and small reconstruction error.
- We successfully deal with missing values.
- We can use the discovered correlations to perform good forecasting, with *much* fewer resources.
- We can easily spot outliers.
- Processing time per stream is constant.

Section 13.6 elaborates on performance and accuracy.

13.5.1 Chlorine concentrations

Description The `Chlorine` dataset was generated by EPANET 2.0[3] that accurately simulates the hydraulic and chemical phenomena within drinking water distribution systems. Given a network as the input, EPANET tracks the flow of water in each pipe, the pressure at each node, the height of water in each tank, and the concentration of a chemical species throughout the network, during a simulation period comprised of multiple timestamps. We monitor the chlorine concentration level at all the 166 junctions in the network shown in Figure 13.3(a), for 4310 timestamps during 15 days (one time tick every five minutes). The data was generated by using the input network with the demand patterns, pressures, flows specified at each node.

Data characteristics The two key features are:

- A clear global periodic pattern (daily cycle, dominating residential demand pattern). Chlorine concentrations reflect this, with few exceptions.

[3] `http://www.epa.gov/ORD/NRMRL/wswrd/epanet.html`

- A slight time shift across different junctions, which is due to the time it takes for fresh water to flow down the pipes from the reservoirs.

Thus, most streams exhibit the same sinusoidal-like pattern, except with gradual phase shifts as we go further away from the reservoir.

Results of SPIRIT SPIRIT can successfully summarise the data using just two numbers (hidden variables) per time tick, as opposed to the original 166 numbers. Figure 13.3(a) shows the reconstruction for four of the sensors (out of 166). Only two hidden variables give very good reconstruction.

Interpretation The two hidden variables (Figure 13.3(b)) reflect the two key dataset characteristics:

- The first hidden variable captures the global, periodic pattern.
- The second one also follows a very similar periodic pattern, but with a slight "phase shift." It turns out that the two hidden variables together are sufficient to express (via a linear combination) any other time series with an arbitrary "phase shift."

13.5.2 Light measurements

Description The `Motes` dataset consists of light intensity measurements collected using Berkeley Mote sensors, at several different locations in a lab (see Figure 13.4), over a period of a month.

Data characteristics The main characteristics are:

- A clear global periodic pattern (daily cycle).
- Occasional big spikes from some sensors (outliers).

Results of SPIRIT SPIRIT detects four hidden variables (see Figure 13.5). Two of these are intermittent and correspond to outliers, or changes in the correlated trends. We show the reconstructions for some of the observed variables in Figure 13.4(b).

Interpretation In summary, the first two hidden variables (see Figure 13.5) correspond to the global trend and the last two, which are intermittently present, correspond to outliers. In particular:

- The first hidden variable captures the global periodic pattern.
- The interpretation of the second one is again similar to the `Chlorine` dataset. The first two hidden variables together are sufficient to express arbitrary phase shifts.
- The third and fourth hidden variables indicate some of the potential outliers in the data. For example, there is a big spike in the 4th hidden variable at time $t = 1033$, as shown in Figure 13.5. Examining the participation weights $\mathbf{w}_4$ at that timestamp, we can find the corresponding sensors "responsible" for this anomaly, i.e., those sensors whose participation weights have very high magnitude. Among these, the most prominent are sensors 31 and 32. Looking at the actual measurements from these sensors, we see that before time $t = 1033$ they are almost 0. Then, very large increases occur around $t = 1033$, which bring an additional hidden variable into the system.

13.5.3 Room temperatures

Description The `Critter` dataset consists of 8 streams (see Figure 13.10). Each stream comes from a small sensor[4] (aka. Critter) that connects to the joystick port and measures temperature. The sensors were placed in 5 neighbouring rooms. Each time tick represents the average temperature during one minute.

Furthermore, to demonstrate how the correlations capture information about missing values, we repeated the experiment after blanking 1.5% of the values (five blocks of *consecutive* timestamps; see Figure 13.7).

Data characteristics Overall, the dataset does not seem to exhibit a clear trend. Upon closer examination, all sensors fluctuate slightly around a constant temperature (which ranges from 22–27^{o}C, or 72–81^{o}F, depending on the sensor). Approximately half of the sensors exhibit a more similar "fluctuation pattern."

[4] `http://www.ices.cmu.edu/sensornets/`

Results of SPIRIT SPIRIT discovers one hidden variable, which is sufficient to capture the general behaviour. However, if we utilise prior knowledge (such as, e.g., that the pre-set temperature was 23^oC), we can ask SPIRIT to detect trends with respect to that. In that case, SPIRIT comes up with two hidden variables, which we explain later.

SPIRIT is also able to deal successfully with missing values in the streams. Figure 13.7 shows the results on the blanked version (1.5% of the total values in five blocks of *consecutive* timestamps, starting at a different position for each stream) of `Critter`. The correlations captured by SPIRIT's hidden variable often provide useful information about the missing values. In particular, on sensor 8 (second row, Figure 13.7), the correlations picked by the *single* hidden variable successfully capture the missing values in that region (consisting of 270 ticks). On sensor 7, (first row, Figure 13.7; 300 blanked values), the upward trend in the blanked region is also picked up by the correlations. Even though the trend is slightly mis-estimated, as soon as the values are observed again, SPIRIT very quickly gets back to near-perfect tracking.

Interpretation If we examine the participation weights in $\mathbf{w}_1$, the largest ones correspond primarily to streams 5 and 6, and then to stream 8. If we examine the data, sensors 5 and 6 consistently have the highest temperatures, while sensor 8 also has a similar temperature most of the time.

However, if the sensors are calibrated based on the fact that these are building temperature measurements, where we have set the thermostat to 23^oC (73^oF), then SPIRIT discovers two hidden variables (see Figure 13.10). More specifically, if we reasonably assume that we have the prior knowledge of what the temperature *should be* (note that this has nothing to do with the average temperature in the observed data) and want to discover what happens around that temperature, we can subtract it from each observation and SPIRIT will discover patterns and anomalies based on this information. Actually, this is what a human operator would be interested in discovering: "Does the system work as I expect it to?" (based on my knowledge of how it should behave) and "If not, what is wrong?" So, in this case, we indeed discover this information.

- The interpretation of the first hidden variable is similar to that of the original signal: sensors 5 and 6 (and, to a lesser extent, 8) deviate from that temperature the most, for most of the time. Maybe the thermostats are broken or set wrong?

- For $\mathbf{w}_2$, the largest weights correspond to sensors 1 and 3, then to 2 and 4. If we examine the data, we notice that these streams follow a similar, fluctuating trend (close

to the pre-set temperature), the first two varying more violently. The second hidden variable is added at time $t = 2016$. If we examine the plots, we see that, at the beginning, most streams exhibit a slow dip and then ascent (e.g., see 2, 4 and 5 and, to a lesser extent, 3, 7 and 8). However, a number of them start fluctuating more quickly and violently when the second hidden variable is added.

13.5.4 River gauges

Description The dataset was collected from the USACE current river conditions website[5]. It consists of river stage (or, water level) data from three different measuring stations in the same river system (see Figure 13.8).

Data characteristics The data exhibit one common trend and has plenty of missing values (26% of all values, for all three streams).

Results and interpretation Examining the three hidden variable weights found by SPIRIT, these have ratios $1.5 : 1.1 : 1$. Indeed, if we look at all 20,000 time ticks, this is what we see; all streams are very similar (since they are from the same river), with the "amplitude" of the fluctuations having roughly these proportions. Hence, one hidden variable is sufficient, the three weights compactly describe the key information and the interpretation is intuitive.

Besides recovering missing values from underlying correlations captured by the few hidden variables, SPIRIT's tracking abilities are not affected even in extreme cases.

13.6 Performance and accuracy

In this section we discuss performance issues. First, we show that SPIRIT requires very limited space and time. Next, we elaborate on the accuracy of SPIRIT's incremental estimates.

13.6.1 Time and space requirements

Figure 13.9 shows that SPIRIT scales linearly with respect to number of streams n and number of hidden variables k. AR per stream and MUSCLES are essentially off the charts from the very beginning. Furthermore, SPIRIT scales linearly with stream size (i.e., requires constant processing time per tuple).

[5] `http://wmw.lrp.usace.army.mil/current/`

The plots were generated using a synthetic dataset that allows us to precisely control each variable. The datasets were generated as follows:

- Pick the number k of trends and generate sine waves with different frequencies, say $y_{t,i} = \sin(2\pi i/kt), 1 \leq i \leq k$. Thus, all trends are pairwise linearly independent.
- Generate each of the n streams as random linear combinations of these k trend signals.

This allows us to vary k, n and the length of the streams at will. For each experiment shown, one of these parameters is varied and the other two are held fixed. The numbers in Figure 13.9 are wall-clock times of our Matlab implementation. Both AR-per-stream as well as MUSCLES (also in Matlab) are several orders of magnitude slower and thus omitted from the charts.

It is worth mentioning that we have also implemented the SPIRIT algorithms in a real system [SPF05c], which can obtain measurements from sensor devices and display hidden variables and trends in real-time.

13.6.2 Accuracy

In terms of accuracy, everything boils down to the quality of the summary provided by the hidden variables. To this end, we show the reconstruction $\tilde{\mathbf{x}}_t$ of $\mathbf{x}_t$, from the hidden variables $\mathbf{y}_t$ in Figure 13.6. One line uses the true principal directions, the other the SPIRIT estimates (i.e., weight vectors). SPIRIT comes very close to repeated PCA.

We should note that this is an unfair comparison for SPIRIT, since repeated PCA requires (i) storing *all* stream values, and (ii) performing a very expensive SVD computation for *each* time tick. However, the tracking is still very good. This is always the case, provided the corresponding eigenvalue is large enough and fairly well-separated from the others. If the eigenvalue is small, then the corresponding hidden variable is of no importance and we do not track it anyway.

Reconstruction error Table 13.3 shows the reconstruction error, $\sum \|\tilde{\mathbf{x}}_t - \mathbf{x}_t\|^2 / \sum \|\mathbf{x}_t\|^2$, achieved by SPIRIT. In every experiment, we set the energy thresholds to $[f_E, F_E] = [0.95, 0.98]$. Also, as pointed out before, we set $\lambda = 0.96$ as a reasonable default value to deal with non-stationarities that may be present in the data, according to recommendations in the literature [Hay92]. Since we want a metric of overall quality, the MSE rate weighs each observation equally and does not take into account the forgetting factor λ.

Dataset	Chlorine	Critter	Motes
MSE rate (SPIRIT)	0.0359	0.0827	0.0669
MSE rate (repeated PCA)	0.0401	0.0822	0.0448

Table 13.3: Reconstruction accuracy (mean squared error rate).

Still, the MSE rate is very close to the bounds we set. In Table 13.3 we also show the MSE rate achieved by repeated PCA. As pointed out before, this is already an unfair comparison. In this case, we set the number of principal components k to the maximum that SPIRIT uses at any point in time. This choice favours repeated PCA even further. Despite this, the reconstruction errors of SPIRIT are close to the ideal, while using orders of magnitude less time and space.

13.7 Conclusion

We focus on finding patterns, correlations and hidden variables, in a large number of streams. Our proposed method has the following desirable characteristics:

- It discovers underlying correlations among multiple streams, incrementally and in real-time [SPF05c] and provides a very compact representation of the stream collection, via a few *hidden variables*.

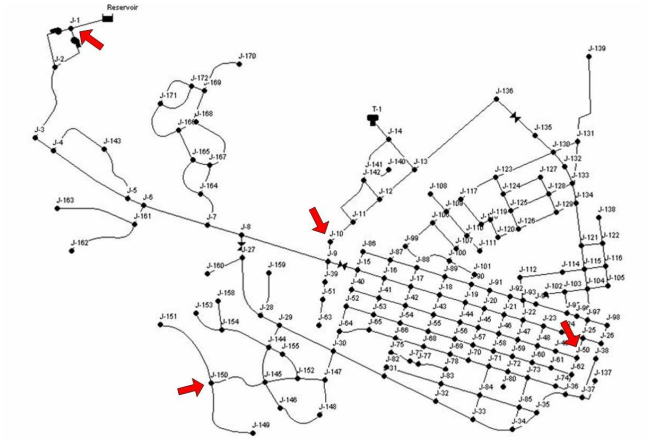
- It automatically estimates the number k of hidden variables to track, and it can automatically adapt, if k changes (e.g., an air-conditioner switching on, in a temperature sensor scenario).

- It scales up extremely well, both on database size (i.e., number of time ticks t), and on the number n of streams. Therefore it is suitable for a large number of sensors / data sources.

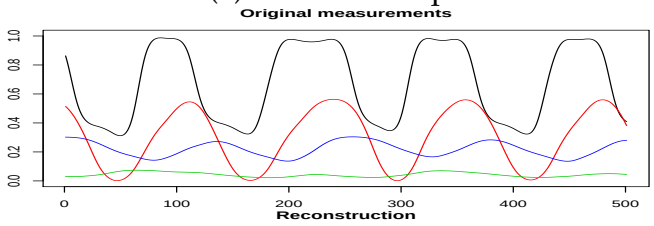
- Its computation demands are low: it only needs $O(nk)$ floating point operations—no matrix inversions nor SVD (both infeasible in online, any-time settings). Its space demands are similarly limited.

- It can naturally hook up with any forecasting method, and thus easily do prediction, as well as handle missing values.

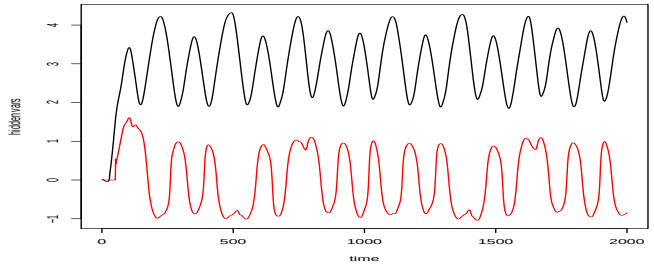
We showed that the output of SPIRIT has a natural interpretation. We evaluated our method on several datasets, where indeed it discovered the hidden variables. Moreover, SPIRIT-based forecasting was several times faster than other methods.



(a) Network map

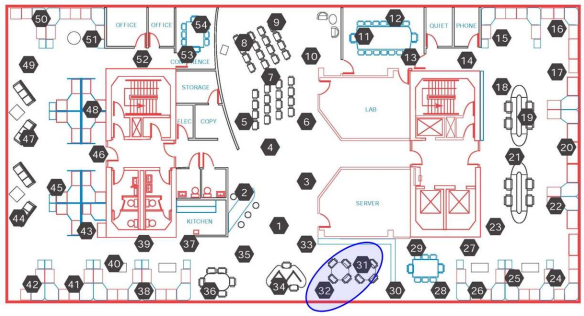


(b) Measurements and SPIRIT reconstruction

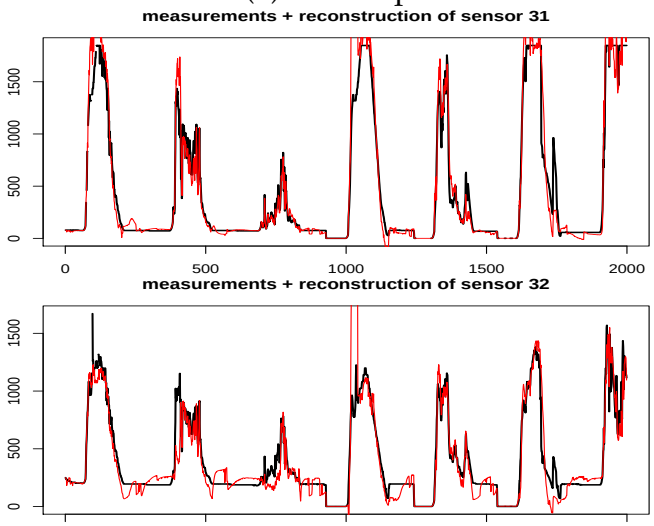


(c) Hidden variables

Figure 13.3: `Chlorine` dataset: (a) the network layout. (b) actual measurements and reconstruction at four junctions (highlighted in (a)). We plot only 500 consecutive timestamps (the patterns repeat after that). (c) shows SPIRIT's hidden variables.



(a) Lab map



(b) Original measurements vs. reconstruction

Figure 13.4: `Mote` dataset: (b) shows the measurements (bold) and reconstruction (thin) on node 31 and 32 (highlighted in (a)).

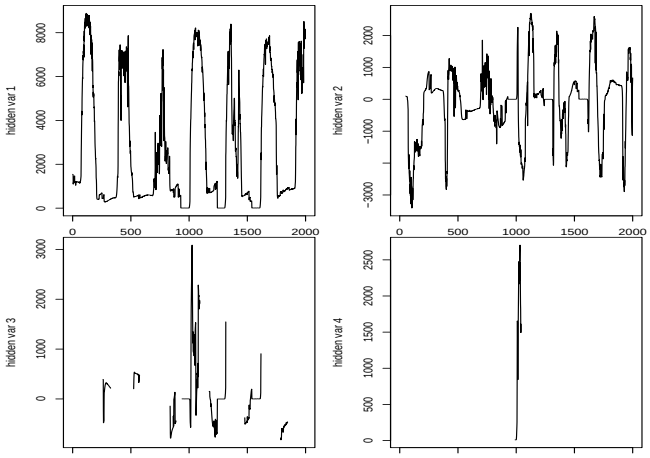


Figure 13.5: `Mote` dataset, hidden variables: The third and fourth hidden variables are intermittent and indicate "anomalous behaviour." Note that the axes limits are different in each plot.

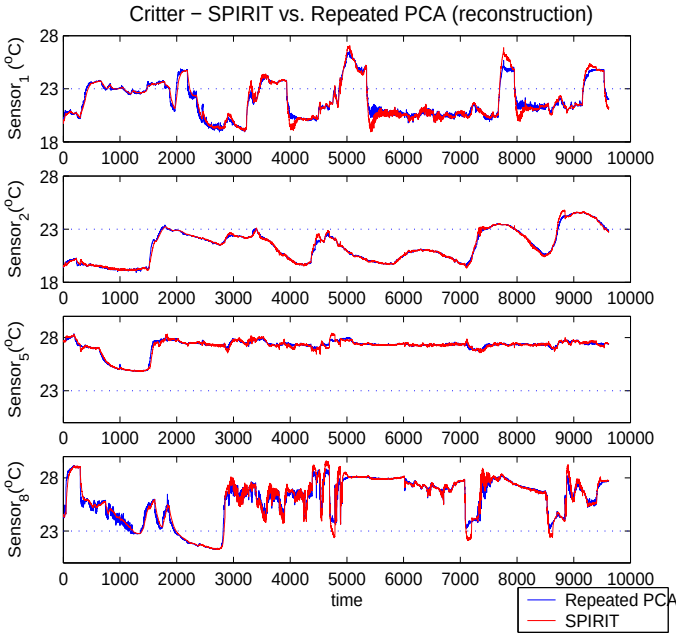


Figure 13.6: Reconstructions $\tilde{\mathbf{x}}_t$ for `Critter`. Repeated PCA requires (i) storing the entire data and (ii) performing PCA at each time tick (quadratic time, at best—for example, wall clock times here are 1.5 minutes versus 7 seconds).

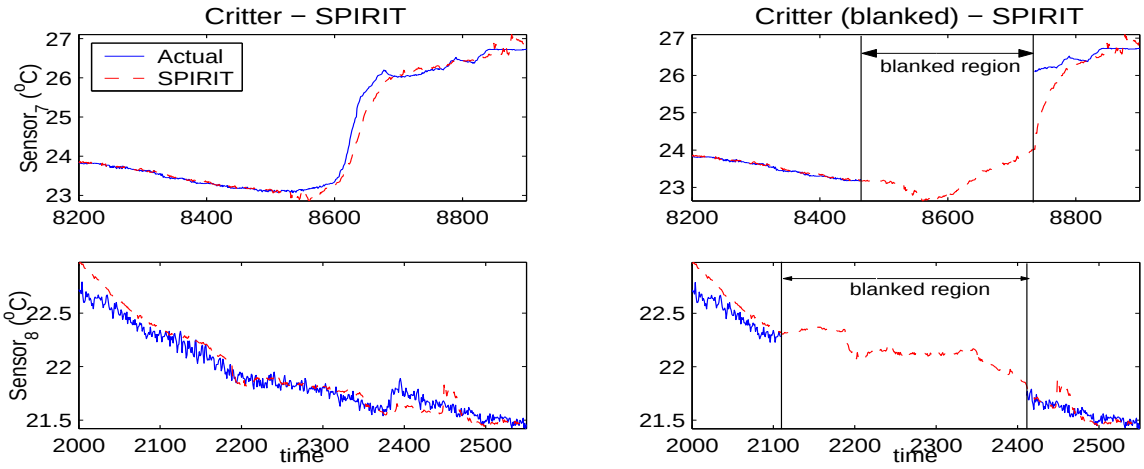
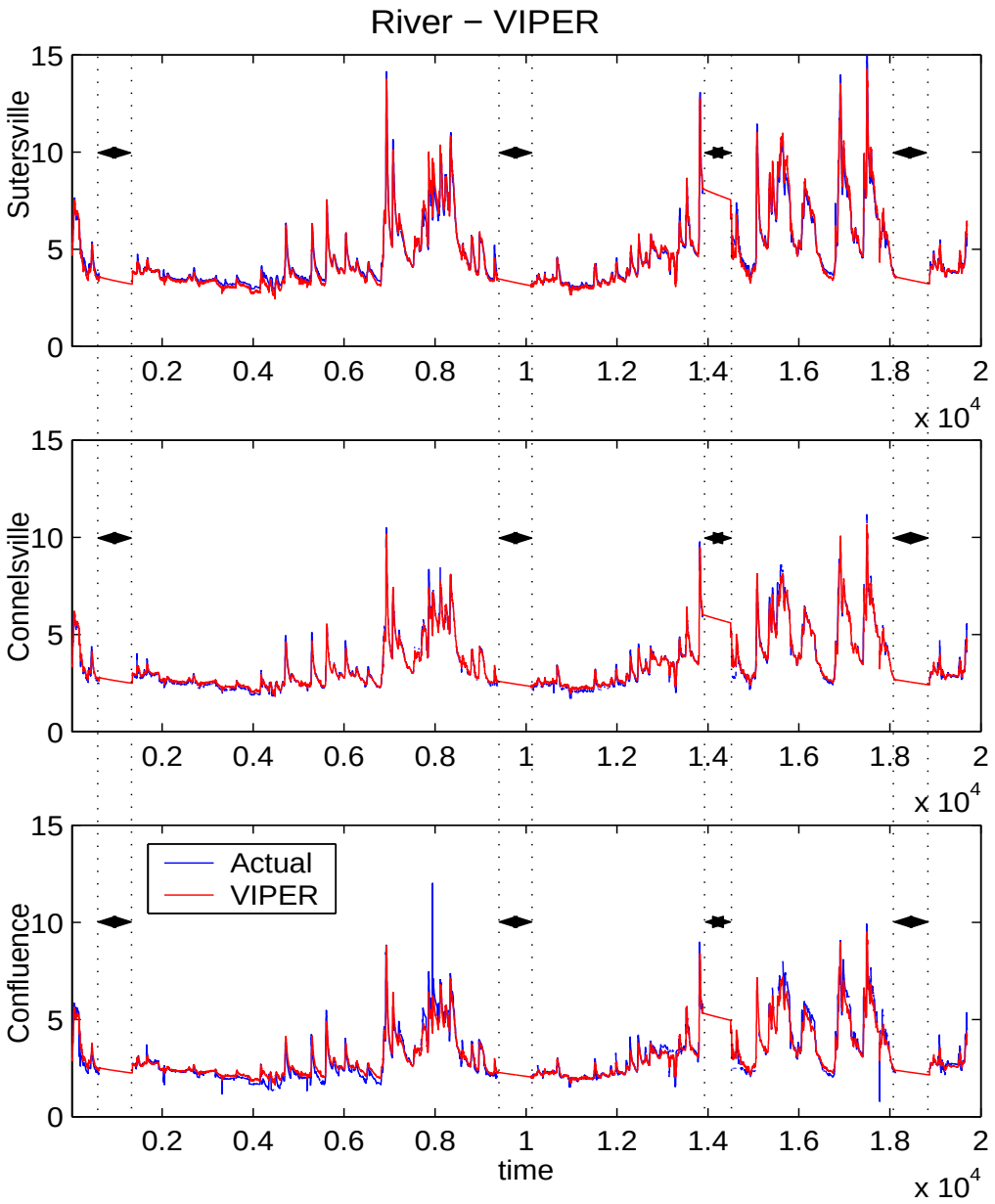
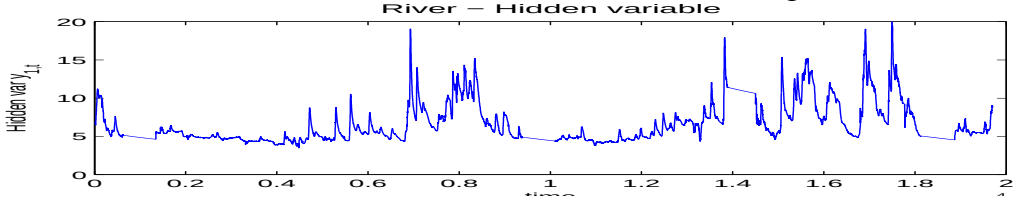


Figure 13.7: Detail of the forecasts on `Critter` with blanked values. The second row shows that the correlations picked by the *single* hidden variable successfully capture the missing values in that region (consisting of 270 *consecutive* ticks). In the first row (300 consecutive blanked values), the upward trend in the blanked region is also picked up by the correlations to other streams. Even though the trend is slightly mis-estimated, as soon as the values are observed again SPIRIT quickly gets back to near-perfect tracking.

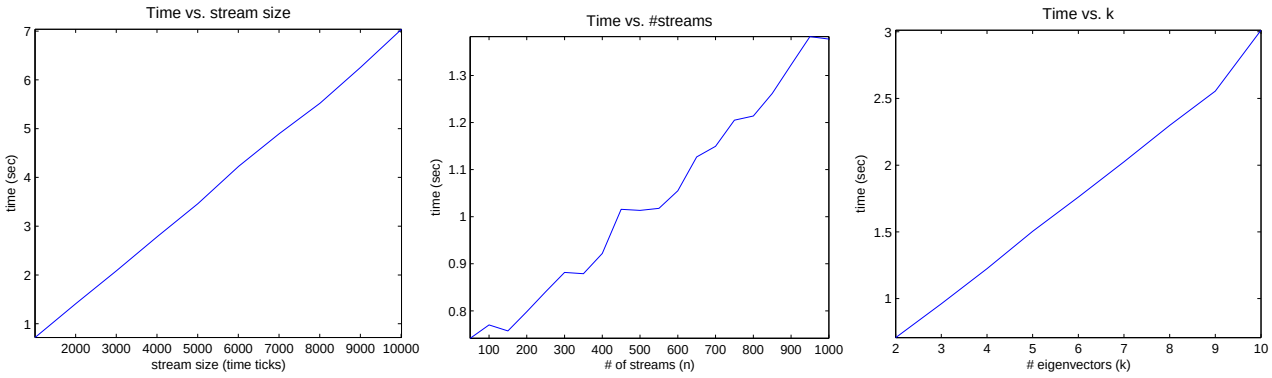


(a) SPIRIT reconstruction (with forecasting)



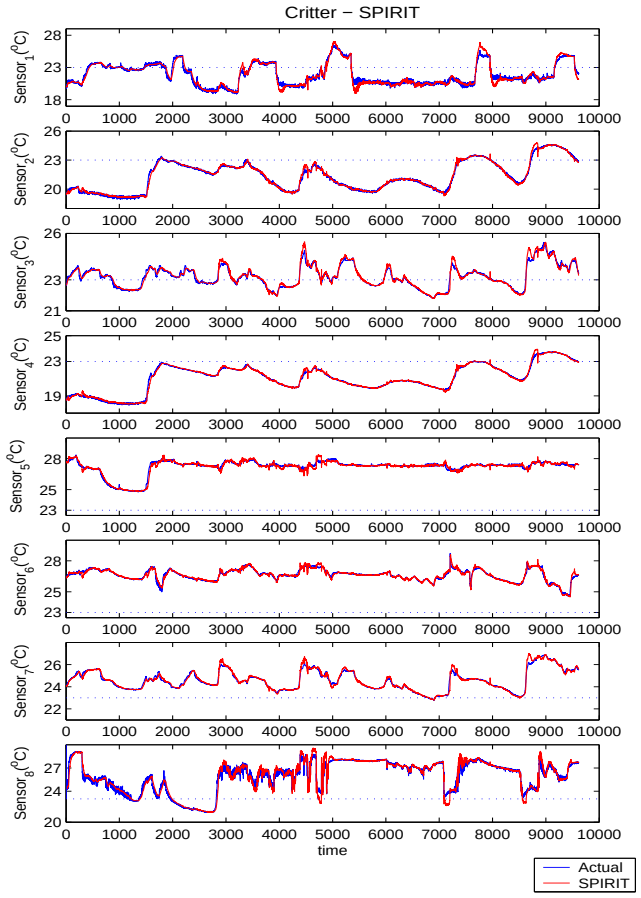
(b) Hidden variable

Figure 13.8: Actual `River` data (river gauges, in feet) and SPIRIT output, for each of the streams (no pun intended). The large portions with missing values across all streams are marked with dotted lines (there are also other missing values in some of the streams); about 26% of all values are missing, but this does not affect SPIRIT's tracking abilities.

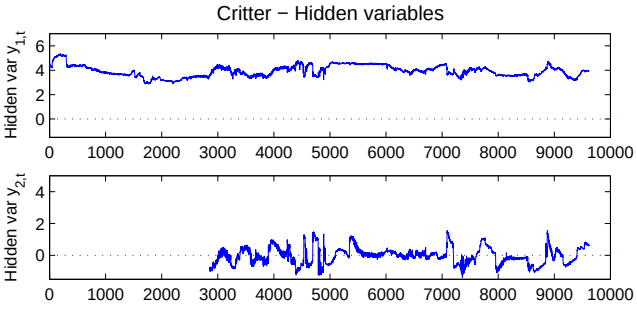


(a) stream size t (b) streams n (c) hidden variables k

Figure 13.9: Wall-clock times (including time to update forecasting models). Times for AR and MUSCLES are not shown, since they are off the charts from the start (13.2 seconds in (a) and 209 in (b)). The starting values are: (a) 1000 time ticks, (b) 50 streams, and (c) 2 hidden variables (the other two held constant for each graph). It is clear that SPIRIT scales linearly.



(a) SPIRIT reconstruction



(b) Hidden variables

Figure 13.10: Actual `Critter` data and SPIRIT output (a), for each of the temperature sensors. The experiment shows that with only two hidden variable, SPIRIT can track the overall behaviour of the entire stream collection. (b) shows the hidden variables.

Chapter 14

Summary

In this part, we considered numerical, time series streams (see Definition 1.5) and develop methods to capture trends at multiple time scales on a single stream, as well as correlations among multiple streams.

In Chapter 12 we presented *AWSOM (Arbitrary Window Stream mOdeling Method)* [PBF03] which allows us to make long range forecasts using limited resources. It allows us to efficiently and effectively discover interesting patterns and trends. This can be done automatically, i.e., with no prior inspection of the data or any user intervention and expert tuning before or during data gathering. Our algorithms require limited resources and can thus be incorporated in sensors—possibly alongside a distributed query processing engine [CCC+02, BGS01, MSHR02]. Updates are performed in constant time with respect to stream size, using logarithmic space. Existing forecasting methods (SARIMA, GARCH, etc) or "traditional" Fourier and wavelet analysis fall short on one or more of these requirements. To the best of our knowledge, AWSOM is the first framework that combines all of the above characteristics.

In Chapter 13 we presented *SPIRIT (Streaming Pattern dIscoveRy in multIple Time-series)* [PSF05]. Given n numerical data streams, all of whose values we observe at each time tick t, SPIRIT can incrementally find correlations and hidden variables, which summarise the key trends in the entire stream collection. It can do this quickly, with no buffering of stream values and without comparing pairs of streams. Moreover, it is any-time, single pass, and it dynamically detects changes. The discovered trends can also be used to immediately spot potential anomalies, to do efficient forecasting and, more generally, to dramatically simplify further data processing.

Chapter 15

Epilogue

In this thesis we develop spatial and stream mining tools for discovery of interesting patterns. These patterns summarise the data, enable forecasting of future trends and spotting of anomalies or outliers. Beyond the emphasis on efficiency and scalability, we focus on simplifying or eliminating user intervention. Data mining algorithms must make the discovery task easy for average users. Unfortunately, many of the existing techniques require non-trivial user intervention at several steps of the process. Eliminating the requirement for user intervention should be a top priority in designing data mining methods.

We developed tools for outlier detection, clustering and forecasting on spatial data (both homogeneous and heterogeneous) and on streams satisfy the following requirements:

- Parameter free: Our methods can search the space of patterns without requiring human intervention and guidance. Any parameters present are data driven and our algorithms produce meaningful patterns when these parameters are set to default values. We also propose techniques to adapt these parameters, when and if user feedback is desired. In any case, users are not exposed to any hard to choose, data dependent parameters. Finally, we also employ the MDL principle to guide the search for the best model.

- Expressive: Our models provide concise, powerful and intuitively interpretable patterns.

- Scalable and any-time: Our algorithms scale to very large datasets. They should require only one pass over the data and in the case of streams, we can incrementally update the models and provide up-to-date patterns instantaneously, while using limited memory.

We show that *multi-resolution* analysis (i.e., examining the data at multiple resolutions or scales) is a powerful tool towards these goals. In particular, for spatial data we employ the *correlation integral*. For time series streams we use the *wavelet transform* and related techniques. Furthermore, we leverage tools from signal processing (again wavelets and, also, *subspace tracking* algorithms) to extract patterns from streams. Finally, we also employ compression principles coupled with multi-level partitions to automatically cluster spatial data. Next, we briefly summarise each part of this thesis.

Part I In this part we investigated how to do outlier detection and clustering on spatial data (see Definition 4) by examining the data at multiple distance scales. First, in Chapter 3, we presented our basic outlier detection scheme, which can detect meaningful outliers based on pairwise distances. In Chapter 4 we show how to extend this scheme to incorporate user feedback when desired, allowing even further data driven exploration, without exposing the users to any parameters. Finally, in Chapter 5 we show how to employ similar ideas for clustering the data based on notions of both *local density* as well as *local dimensionality*. The main proposed technique is a parameter-free transformation of high-dimensional points into a pair of local density and dimensionality.

Part II In this part we explore further analyses possible on the extended spatial data model (see Definition 5). First, in Chapter 8, we investigate the implications of having binary class labels associated with each data point for outlier detection and we introduce *cross-outliers*. In Chapter 9 we explore how to automatically and simultaneously find spatial co-location and binary feature co-occurrence patterns, when an arbitrary number of binary features (or, attributes) are associated with each point. To this end, we employ the Minimum Description Length (MDL) principle, which is a powerful tool for automatic model selection.

Part III In this part we present data mining methods for stream data (see Definition 6). In Chapter 12 we show how to incrementally capture trends at multiple time scales, using limited memory and CPU resources. In Chapter 13 we examine *multiple* streams and consider the problem of capturing correlations and finding hidden variables corresponding to trends on collections of time series streams. These correlations can be used to do efficient forecasting and, consequently, detect outliers along the time dimension. Furthermore, changes in number of correlated trends can be used to detect anomalies.

15.1 Discussion

In this section we discuss potential future directions, first for spatial and then for stream data.

15.1.1 Spatial mining

Cross-outlier detection with multiple classes In Chapter 8 we considered the case of two classes of points. Extending this to multiple classes is an interesting and challenging problem, which is also related to frequent itemset mining as well as algorithms for spatial co-location pattern mining, based on frequent itemsets. However, most existing approaches require significant user guidance.

Extensions to simultaneous spatial and feature clustering In Chapter 9 we dealt with raster data. In several cases, data are naturally available in this form (e.g., per city block or per observation site/patch). Extending the to vector data is another interesting problem. Also, another interesting direction is to explore other forms of data that have some notion of "neighbours," such as web graphs (where "proximity" would be defined in terms of hyperlink distance, rather than geographical location—although information related to geographical location or network topology might also be incorporated).

15.1.2 Stream mining

Correlations and multiple timescale forecasting Combining the methods presented in Chapter 12 and 13 into a unified tool is an interesting but non-trivial task. For example, it is not clear whether applying AWSOM to the correlations discovered at the finest time resolution is sufficient, or whether it would make sense to do correlation discovery per frequency band. For example, in automobile traffic, the presence of highly irregular "noise" at small time scales would probably prevent discovery of correlations. However, the daily time scale, most sensors should be highly correlated to each other.

Lag correlations SPIRIT (Chapter 13) considers correlations among streams at a single time slice: values of different streams are "compared" to discover correlations only if these values belong to the same timestamp. However, in certain cases, two or more streams may be *lag correlated*, i.e., they may look similar if they are delayed (or lagged) for a certain amount of time ticks. For example, sensors measuring traffic along a highway will show

the same trend, but the measurements "downstream" will be delayed by the distance between the sensors divided by the average travel speed. We have already examined this problem in [SPF05a, SPF05b], but further improvements in efficiency may be possible. Also, the algorithms presented in this thesis could potentially also be incorporated in this mining process.

Structural periodicity detection In this case, we wish to discover the dominant periods present in a time series stream. If the periods are known or if the series are of finite length, the solution is well-known. However, performing the same task in a streaming environment with limited resources poses interesting challenges. Furthermore, this problem is related to lag correlations, since a periodic time series is, by definition, lag correlated with itself with the lag equal to the period.

Distributed mining In Chapter 13 we consider the case where the measurements from all streams are collected at a central node. It would be interesting to explore how to perform the same task in a distributed fashion, by discovering trends and hidden variables at a local, neighbourhood level and then communicating this information as necessary, possibly in a hierarchical fashion, in order to discover global correlations. Furthermore, the summary consisting of the hidden variables (without the participation weights) may be viewed as a form of anonymising the individual measurements, thus protecting privacy.

Bibliography

[AAR96] A. Arning, R. Agrawal, and P. Raghavan. A linear method for deviation detection in large database. In *Proc. KDD*, pages 164–169, 1996.

[ABB+02] Arvind Arasu, Brian Babcock, Shivnath Babu, Jon McAlister, and Jennifer Widom. Characterizing memory requirements for queries over continuous data streams. In *PODS*, 2002.

[ABKS99] Mihael Ankerst, Markus M. Breunig, Hans-Peter Kriegel, and Jörg Sander. OPTICS: Ordering points to identify the clustering structure. In *Proc. SIGMOD*, 1999.

[ACC+03] Daniel J. Abadi, Don Carney, Ugur Cetintemel, Mitch Cherniack, Christian Convey, Sangdon Lee, Michael Stonebraker, Nesime Tatbul, and Stan Zdonik. Aurora: a new model and architecture for data stream management. *The VLDB Journal*, 12(2):120–139, 2003.

[Agg03] Charu C. Aggarwal. A framework for diagnosing changes in evolving data streams. In *SIGMOD*, 2003.

[AGGR98] Rakesh Agrawal, Johannes Gehrke, Dimitrios Gunopulos, and Prabhakar Raghavan. Automatic subspace clustering of high dimensional data for data mining applications. In *Proc. SIGMOD*, 1998.

[AHY03] Charu C. Aggarwal, Jiawei Han, and Philip S. Yu. A framework for clustering evolving data streams. In *VLDB*, 2003.

[Aka97] Metin Akay, editor. *Time Frequency and Wavelets in Biomedical Signal Processing*. J. Wiley, 1997.

[AM04] Pankaj K. Agarwal and Nabil H. Mustafa. k-means projective clustering. In *PODS*, 2004.

[AMAK05] M.H. Ali, Mohamed F. Mokbel, Walid Aref, and Ibrahim Kamel. Detection and tracking of discrete phenomena in sensor network databases. In *SSDBM*, 2005.

[AMN+98] Sunil Arya, David M. Mount, Nathan S. Netanyahu, Ruth Silverman, and Angela Y. Wu. An optimal algorithm for approximate nearest neighbor searching fixed dimensions. *JACM*, 45(6), 1998.

[APW+99] Charu C. Aggarwal, Cecilia Magdalena Procopiuc, Joel L. Wolf, Philip S. Yu, and Jong Soo Park. Fast algorithms for projected clustering. In *Proc. SIGMOD*. ACM, 1999.

[AS94] Rakesh Agrawal and Ramakrishnan Srikant. Fast algorithms for mining association rules in large databases. In *Proc. 20th VLDB*, pages 487–499, 1994.

[ATMS04] Periklis Andritsos, Panayiotis Tsaparas, Reneé Miller, and Ken Sevcik. LIMBO: Scalable clustering for categorical data. In *EDBT*, 2004.

[AY00] Charu C. Aggarwal and Philip S. Yu. Finding generalized projected clusters in high dimensional spaces. In *Proc. SIGMOD*. ACM, 2000.

[AY01] C.C. Aggarwal and P.S. Yu. Outlier detection for high dimensional data. In *Proc. SIGMOD*, 2001.

[BBKK97] Steffan Berchtold, Christisn Böhm, Daniel A. Keim, and Hans-Peter Kriegel. A cost model for nearest neighbor search in high-dimensional data space. In *Proc. PODS*, pages 78–86, 1997.

[BBM04] Sugato Basu, Mikhail Bilenko, and Raymond J. Mooney. A probabilistic framework for semi-supervised clustering. In *Proc. KDD*, 2004.

[BC00] Daniel Barbará and Ping Chen. Using the fractal dimension to cluster datasets. In *Proc. KDD*, pages 260–264, 2000.

[BD91] Peter J. Brockwell and Richard A. Davis. *Time Series: Theory and Methods*. Springer Series in Statistics. Springer-Verlag, 2nd edition, 1991.

[BDM02] Brian Babcock, Mayur Datar, and Rajeev Motwani. Sampling from a moving window over streaming data. In *SODA*, 2002.

[Ber93] M. Bern. Approximate closest-point queries in high dimension. *Information Processing Letters*, 45:95–99, 1993.

[Ber94] Jean Beran. *Statistics for Long-Memory Processes*. Chapman & Hall, 1994.

[BF95] A. Belussi and C. Faloutsos. Estimating the selectivity of spatial queries using the 'correlation' fractal dimension. In *Proc. VLDB*, pages 299–310, 1995.

[BF98] Alberto Belussi and Christos Faloutsos. Self-spacial join selectivity estimation using fractal concepts. *ACM TOIS*, 16(2), 1998.

[BGS01] Philippe Bonnet, Johannes E. Gehrke, and Praveen Seshadri. Towards sensor database systems. In *Proc. MDM*, 2001.

[BKK96] Stefan Berchtold, Daniel A. Keim, and Hans-Peter Kriegel. The X-tree: An index structure for high-dimensional data. In *Proc. VLDB*, 1996.

[BKNS00] M.M. Breunig, H.P. Kriegel, R.T. Ng, and J. Sander. Lof: Identifying density-based local outliers. In *Proc. SIGMOD Conf.*, pages 93–104, 2000.

[BL94] V. Barnett and T. Lewis. *Outliers in Statistical Data*. John Wiley, 1994.

[BO03] Brian Babcock and Chris Olston. Distributed top-k monitoring. In *Proc. SIGMOD*, 2003.

[Bol86] Tim Bollerslev. Generalized autoregressive conditional heteroskedasticity. *J. Econometrics*, 31:307–327, 1986.

[BS03a] S. D. Bay and M. Schwabacher. Mining distance-based outliers in near linear time with randomization and a simple pruning rule. In *Proc. KDD*, pages 24–7, 2003.

[BS03b] Ahmet Bulut and Ambuj K. Singh. SWAT: Hierarchical stream summarization in large networks. In *Proc. 19th ICDE*, 2003.

[CCC+02] Donald Carney, Ugur Cetintemel, Mitch Cherniack, Christian Convey, Sangdon Lee, Greg Seidman, Michael Stonebraker, Nesime Tatbul, and Stanley B. Zdonik. Monitoring streams – a new class of data management applications. In *Proc. VLDB*, 2002.

[CCD+03] Sirish Chandrasekaran, Owen Cooper, Amol Deshpande, Michael J. Franklin, Joseph M. Hellerstein, Wei Hong, Sailesh Krishnamurthy, Samuel Madden, Vijayshankar Raman, Fred Reiss, and Mehul A. Shah. TelegraphCQ: Continuous dataflow processing for an uncertain world. In *CIDR*, 2003.

[CCR+03] Donald Carney, Ugur Cetintemel, Alex Rasin, Stanley B. Zdonik, Mitch Cherniack, and Michael Stonebraker. Operator scheduling in a data stream manager. In *VLDB*, 2003.

[CDH+02] Yixin Chen, Guozhu Dong, Jiawei Han, Benjamin W. Wah, and Jianyong Wang. Multi-dimensional regression analysis of time-series data streams. In *Proc. VLDB*, 2002.

[CDIM02] Graham Cormode, Mayur Datar, Piotr Indyk, and S. Muthukrishnan. Comparing data streams using hamming norms (how to zero in). In *VLDB*, 2002.

[CGN00] L.Ñichard Carley, Gregory R. Ganger, and David Nagle. Mems-based integrated-circuit mass-storage systems. *CACM*, 43(11):72–80, 2000.

[CJSS03] Chuck Cranor, Theodore Johnson, Oliver Spataschek, and Vladislav Shkapenyuk. Gigascope: a stream database for network applications. In *SIGMOD*, 2003.

[CLKB04] Jeffrey Considine, Feifei Li, George Kollios, and John W. Byers. Approximate aggregation techniques for sensor databases. In *Proc. ICDE*, 2004.

[CNBYM01] E. Chávez, G. Navarro, R. Baeza-Yates, and J.L. Marroquín. Searching in metric spaces. *ACM Comp. Surveys*, 33(3):273–321, 2001.

[CPMF04] Deepayan Chakrabarti, Spiros Papadimitriou, Dharmendra Modha, and Christos Faloutsos. Fully automatic cross-associations. In *Proc. KDD*, 2004.

[CPZ97] Paolo Ciaccia, Marco Patella, and Pavel Zezula. M-tree: An efficient access method for similarity search in metric spaces. In *Proc. VLDB*, 1997.

[CT91] Thomas M. Cover and Joy A. Thomas. *Elements of Information Theory*. Wiley-Interscience, 1991.

[DGGR02] Alin Dobra, Minos N. Garofalakis, Johannes Gehrke, and Rajeev Rastogi. Processing complex aggregate queries over data streams. In *Proc. SIGMOD*, 2002.

[DGI+02] Mayur Datar, Aristides Gionis, Piotr Indyk, , and Rajeev Motwani. Maintaining stream statistics over sliding windows. In *Proc. SODA*, 2002.

[DGMH05] Amol Deshpande, Carlos Guestrin, Samuel Madden, and Wei Hong. Exploiting correlated attributes in acqusitional query processing. In *ICDE*, 2005.

[DGR03] Abhinandan Das, Johannes Gehrke, and Mirek Riedewald. Approximate join processing over data streams. In *SIGMOD*, 2003.

[DH00] Pedro Domingos and Geoff Hulten. Mining high-speed data streams. In *KDD*, 2000.

[DK96] Kostas I. Diamantaras and Sun-Yuan Kung. *Principal Component Neural Networks: Theory and Applications*. John Wiley, 1996.

[DLM+98] Gautam Das, King-Ip Lin, Heikki Mannila, Gopal Renganathan, and Padhraic Smyth. Rule discovery from time series. In *Proc. KDD*, 1998.

[DM01] Inderjit S. Dhillon and Dharmendra S. Modha. Concept decompositions for large sparse text data using clustering. *Mach. Learning*, 42:143–175, 2001.

[DMM03] Inderjit S. Dhillon, Subramanyam Mallela, and Dharmendra S. Modha. Information-theoretic co-clustering. In *KDD*, 2003.

[DS02] Morris H. DeGroot and Mark J. Schervish. *Probability and Statistics*. Addison-Wesley, 3rd edition, 2002.

[EKSX96] Martin Ester, Hans-Peter Kriegel, Jörg Sander, and Xiaowei Xu. A density-based algorithm for discovering clusteris in large spatial databases with noise. In *Proc. KDD*, 1996.

[EMS04] Funda Ergün, S. Muthukrishnan, and Süleyman Cenk Sahinalp. Sublinear methods for detecting periodic trends in datastreams. In *Proc. LATIN*, 2004.

[Fal96] Christos Faloutsos. *Searching Multimedia Databases by Content*. Kluwer Academic Inc., 1996.

[FK94] Christos Faloutsos and Ibrahim Kamel. Beyond uniformity and independence: Analysis of r-trees using the concept of fractal dimension. In *Proc. PODS*, 1994.

[FLM77] T. Fiegel, J. Lindenstrauss, and V.D. Millman. The dimensions of almost spherical sections of convex bodies. *Acta Math.*, 139(1-2):53–94, 1977.

[FSJT00] Christos Faloutsos, Bernhard Seeger, Caetano Traina Jr., and Agma Traina. Spatial join selectivity using power laws. In *Proc. SIGMOD*, pages 177–188, 2000.

[FTCTF01] Roberto F. Santos Filho, Agma J. M. Traina, Jr. Caetano Traina, and Christos Faloutsos. Similarity search without tears: The omni family of all-purpose access methods. In *Proc. ICDE*, 2001.

[Fuk90] Keinosuke Fukunaga. *Introduction to Statistical Pattern Recognition*. Academic Press, 1990.

[GG02] Minos N. Garofalakis and Phillip B. Gibbons. Wavelet synopses with error guarantees. In *Proc. SIGMOD*, 2002.

[GGK03] Sudipto Guha, Dimitrios Gunopulos, and Nick Koudas. Correlating synchronous and asynchronous data streams. In *KDD*, 2003.

[GGR02] Venkatesh Ganti, Johannes Gehrke, and Raghu Ramakrishnan. Mining data streams under block evolution. *SIGKDD Explorations*, 3(2):1–10, 2002.

[GGR03] Sumit Ganguly, Minos Garofalakis, and Rajeev Rastogi. Processing set expressions over continuous update streams. In *SIGMOD*, 2003.

[GHPT05] Aristides Gionis, Alexander Hinneburg, Spiros Papadimitriou, and Panayiotis Tsaparas. Dimension induced clustering. In *Proc. KDD*, 2005.

[GIM99] Aristides Gionis, Piotr Indyk, and Rajeev Motwani. Similarity search in high dimensions via hashing. In *Proc. VLDB*, pages 518–528, 1999.

[GK02] Sudipto Guha and Nick Koudas. Approximating a data stream for querying and estimation: Algorithms and performance evaluation. In *Proc. ICDE*, 2002.

[GKMS01] Anna C. Gilbert, Yannis Kotidis, S. Muthukrishnan, and Martin Strauss. Surfing wavelets on streams: One-pass summaries for approximate aggregate queries. In *Proc. VLDB*, 2001.

[GKS01] Johannes Gehrke, Flip Korn, and Divesh Srivastava. On computing correlated aggregates over continual data streams. In *Proc. SIGMOD*, 2001.

[GKS04] Sudipto Guha, Chulyun Kim, and Kyuseok Shim. XWAVE: Optimal and approximate extended wavelets for streaming data. In *VLDB*, 2004.

[GMM+03] Sudipto Guha, Adam Meyerson, Nina Mishra, Rajeev Motwani, and Liadan O'Callaghan. Clustering data streams: Theory and practice. *IEEE TKDE*, 15(3):515–528, 2003.

[Goo84] Jane Goodall. *The Chimpanzees of Gombe: Patterns of Behaviour*. Harvard Univ. Press, 1984.

[GRS98] Sudipto Guha, Rajeev Rastogi, and Kyuseok Shim. CURE: an efficient clustering algorithm for large databases. In *Proc. SIGMOD*, pages 73–84, 1998.

[Grü05] Peter Grünwald. A tutorial introduction to the minimum description length principle. In *Advances in Minimum Description Length: Theory and Applications*. MIT Press, 2005.

[GSW01] Ramazan Gencay, Faruk Selcuk, and Brandon Whitcher. *An Introduction to Wavelets and Other Filtering Methods in Finance and Economics*. Academic Press, 2001.

[Haw80] D.M. Hawkins. *Identification of Outliers*. Chapman and Hall, 1980.

[Hay92] Simon Haykin. *Adaptive Filter Theory*. Prentice Hall, 1992.

[HE03] Gregory Hamerly and Charles Elkan. Learning the k in k-means. In *Proc. 17th NIPS*, 2003.

[HK98] Alexander Hinneburg and Daniel A. Keim. An efficient approach to clustering in large multimedia databases with noise. In *Proc. 4th KDD*, pages 58–65, 1998.

[HK01] Jiawei Han and Micheline Kamber. *Data Mining: Concepts and Techniques*. Morgan Kaufmann, 2001.

[HPYM04] Jiawei Han, Jian Pei, Yiwen Yin, and Runying Mao. Mining frequent patterns without candidate generation: A frequent-pattern tree approach. *Data Min. Knowl. Discov.*, 8(1):53–87, 2004.

[HSD01] Geoff Hulten, Laurie Spencer, and Pedro Domingos. Mining time-changing data streams. In *KDD*, 2001.

[HSW+00] Jason Hill, Robert Szewczyk, Alec Woo, Seth Hollar, David Culler, and Kristofer Pister. System architecture directions for networked sensors. In *Proc. ASPLOS-IX*, 2000.

[HTF01a] Trevor Hastie, Robert Tibshirani, and Jerome Friedman. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*. Springer, 2001.

[HTF01b] Trevor Hastie, Robert Tibshirani, and Jerome H. Friedman. *The Elements of Statistical Learning: Data Mining, Inference and Prediction*. Springer Verlag, 2001.

[HXSP03] Yan Huang, Hui Xiong, Shashi Shekhar, and Jian Pei. Mining confident co-location rules without a support threshold. In *Proc. SAC*, pages 497–501, 2003.

[IKM00] Piotr Indyk, Nick Koudas, and S. Muthukrishnan. Identifying representative trends in massive time series data sets using sketches. In *Proc. VLDB*, 2000.

[ISC] SANS internet storm center (ISC). `http://isc.sans.org/`.

[JKM99] H.V. Jagadish, Nick Koudas, and S. Muthukrishnan. Mining deviants in a time series database. In *Proc. VLDB*, pages 102–113, 1999.

[JKN98] Theodore Johnson, Ivy Kwok, and Raymond T. Ng. Fast computation of 2-dimensional depth contours. In *Proc. KDD*, pages 224–228, 1998.

[JMF99] A.K. Jain, M.N. Murty, and P.J. Flynn. Data clustering: A review. *ACM Comp. Surveys*, 31(3):264–323, 1999.

[Jol02] I.T̃. Jolliffe. *Principal Component Analysis*. Springer, 2002.

[JTH01] Wen Jin, Anthony .K.H. Tung, and Jiawei Han. Mining top-n local outliers in large databases. In *Proc. KDD*, pages 293–298, 2001.

[KHK99] George Karypis, Eui-Hong Han, and Vipin Kumar. Chameleon: Hierarchical clustering using dynamic modeling. *IEEE Computer*, 32(8):68–75, 1999.

[KK98] George Karypis and Vipin Kumar. Multilevel algorithms for multi-constraint graph partitioning. In *Proc. SC98*, pages 1–13, 1998.

[KL04] Robert Krauthgamer and James R. Lee. The black-box complexity of nearest neighbor search. In *Proc. ICALP*, 2004.

[KLR04a] Eamonn Keogh, Stefano Lonardi, and Chotirat Ann Ratanamahatana. Towards parameter-free data mining. In *KDD*, 2004.

[KLR04b] Eamonn Keogh, Stefano Lonardi, and Chotirat Ann Ratanamahatana. Towards parameter-free data mining. In *KDD*, 2004.

[KN96] Edwin M. Knorr and Raymond T. Ng. Finding aggregate proximity relationships and commonalities in spatial data mining. *IEEE TKDE*, 8(6):884–897, 1996.

[KN97] Edwin M. Knorr and Raymond T. Ng. A unified notion of outliers: Properties and computation. In *Proc. KDD*, pages 219–222, 1997.

[KN98] Edwin M. Knorr and Raymond T. Ng. Algorithms for mining distance-based outliers in large datasets. In *Proc. VLDB 1998*, pages 392–403, 1998.

[KN99] Edwin M. Knorr and Raymond T. Ng. Finding intentional knowledge of distance-based outliers. In *Proc. VLDB*, pages 211–222, 1999.

[KNT00] Edwin M. Knorr, Raymond T. Ng, and Vladimir Tucakov. Distance-based outliers: Algorithms and applications. *VLDB Journal*, 8:237–253, 2000.

[KS97] Norio Katayama and Shin'ichi Satoh. The SR-tree: an index structure for high-dimensional nearest neighbor queries. In *SIGMOD*, 1997.

[KT02] Jon Kleinberg and Éva Tardos. Approximation algorithms for classification problems with pairwise relationships: Metric labeling and Markov random fields. *JACM*, 49(5):616–639, 2002.

[LIB] Libsvm. `http://www.csie.nut.edu.tw/~cjlin/libsvm`.

[LMP03] Antti Leino, Heikki Mannila, and Ritva Liisa Pitkänen. Rule discovery and probabilistic modeling for onomastic data. In *Proc. PKDD*, pages 291–302, 2003.

[LPC+04] Anukool Lakhina, Konstantina Papagiannaki, Mark Crovella, Christophe Diot, Eric D. Kolaczyk, and Nina Taft. Structural analysis of network traffic flows. In *Proc. SIGMETRICS*, 2004.

[LTWW94] W.E. Leland, M.S. Taqqu, W. Willinger, and D.V. Wilson. On the self-similar nature of ethernet traffic. *IEEE Trans. on Networking*, 2(1):1–15, 1994.

[LVKG04] Jessica Lin, Michail Vlachos, Eamonn Keogh, and Dimitrios Gunopulos. Iterative incremental clustering of time series. In *EDBT*, 2004.

[MRS03] Nina Mishra, Dana Ron, and Ram Swaminathan. On finding large conjunctive clusters. In *Proc. 16th COLT*, 2003.

[MSHR02] Samuel R. Madden, Mehul A. Shah, Joseph M. Hellerstein, and Vijayshankar Raman. Continuously adaptive continuous queries over streams. In *SIGMOD Conf.*, 2002.

[MWA+03] Rajeev Motwani, Jennifer Widom, Arvind Arasu, Brian Babcock, Shivnath Babu, Mayur Datar, Gurmeet Manku, Chris Olston, Justin Rosenstein, and Rohit Varma. Query processing, resource management, and approximation in a data stream management system. In *CIDR*, 2003.

[NH94] Raymond T. Ng and Jiawei Han. Efficient and effective clustering methods for spatial data mining. In *Proc. VLDB*, pages 144–155, 1994.

[NJW01] Andrew Y. Ng, Michael I. Jordan, and Y. Weiss. On spectral clustering: Analysis and an algorithm. In *Proc. NIPS*, pages 849–856, 2001.

[Oja89] Erkki Oja. Neural networks, principal components, and subspaces. *Intl. J. Neural Syst.*, 1:61–68, 1989.

[OJW03] Chris Olston, Jing Jiang, and Jennifer Widom. Adaptive filters for continuous queries over distributed data streams. In *Proc. SIGMOD*, 2003.

[PBF03] Spiros Papadimitriou, Anthony Brockwell, and Christos Faloutsos. Adaptive, hands-off stream mining. In *VLDB*, 2003.

[PF03] Spiros Papadimitriou and Christos Faloutsos. Cross-outlier detection. In *Proc. SSTD*, 2003.

[PGT+05] Spiros Papadimitirou, Aristides Gionis, Panayiotis Tsaparas, Heikki Mannila, and Christos Faloutsos. Parameter-free spatial data mining using MDL. In *Proc. ICDM*, 2005.

[PJAM02] Cecilia M. Procopiuc, Michael Jones, Pankaj K. Agarwal, and T. M. Murali. A monte carlo algorithm for fast projective clustering. In *Proc. SIGMOD*. ACM Press, 2002.

[PKF00] Bernd-Uwe Pagel, Flip Korn, and Christos Faloutsos. Deflating the dimensionality curse using multiple fractal dimensions. In *Proc. ICDE*, 2000.

[PKGF03] Spiros Papadimitriou, Hiroyuki Kitagawa, Philip B. Gibbons, and Christos Faloutsos. LOCI: Fast outlier detection using the local correlation integral. In *Proc. ICDE*, 2003.

[PM00] Dan Pelleg and Andrew Moore. X-means: Extending K-means with efficient estimation of the number of clusters. In *Proc. 17th ICML*, pages 727–734, 2000.

[Pot52] Renfrey Burnard Potts. Some generalized order-disorder transformations. *Proc. Camb. Phil. Soc.*, 48:106, 1952.

[PSF05] Spiros Papadimitriou, Jimeng Sun, and Christos Faloutsos. Streaming pattern discovery in multiple time-series. In *Proc. VLDB*, 2005.

[PVK+04] Themistoklis Palpanas, Michail Vlachos, Eamonn J. Keogh, Dimitrios Gunopulos, and Wagner Truppel. Online amnesic approximation of streaming time series. In *Proc. ICDE*, 2004.

[PW00] Donald B. Percival and Andrew T. Walden. *Wavelet Methods for Time Series Analysis*. Cambridge Univ. Press, 2000.

[Ras90] S. Neil Rasband. *Chaotic Dynamics of Nonlinear Systems*. Wiley-Interscience, 1990.

[RFGN00] Erik Riedel, Christos Faloutsos, Gregory R. Ganger, and David Nagle. Data mining on an OLTP system (nearly) for free. In *SIGMOD Conf.*, 2000.

[Ris83] Jorma Rissanen. Universal prior for integers and estimation by minimum description length. *Annals of Statistics*, 11(2):416–431, 1983.

[RK01] P. Krishna Reddy and Masaru Kitsuregawa. An approach to relate the web communities through bipartite graphs. In *Proc. WISE*, pages 302–310, 2001.

[RL79] Jorma Rissanen and Glen G. Langdon Jr. Arithmetic coding. *IBM J. Res. Dev.*, 23:149–162, 1979.

[RL87] P.J. Rousseeuw and A.M. Leroy. *Robust Regression and Outlier Detection*. John Wiley and Sons, 1987.

[Sal04] Marko Salmenkivi. Evaluating attraction in spatial point patterns with an application in the field of cultural history. In *ICDM*, 2004.

[Sch88] H.G. Schuster. *Deterministic Chaos*. VCH Publisher, 1988.

[SCZ98] Gholamhosein Sheikholeslami, Surojit Chatterjee, and Aidong Zhang. WaveCluster: A multi-resolution clustering approach for very large spatial databases. In *Proc. VLDB*, 1998.

[SDS] Sloan digital sky survey (SDSS). `http://www.sdss.org/`.

[SPF05a] Yasushi Sakurai, Spiros Papadimitriou, and Christos Faloutsos. Autolag: Automatic discovery of lag correlations in stream data. In *Proc. ICDE*, 2005.

[SPF05b] Yasushi Sakurai, Spiros Papadimitriou, and Christos Faloutsos. BRAID: Stream mining through group lag correlations. In *SIGMOD*, 2005.

[SPF05c] Jimeng Sun, Spiros Papadimitriou, and Christos Faloutsos. Online latent variable detection in sensor networks. In *Proc. ICDE*, 2005. (demo).

[TCZ+03] Nesime Tatbul, Ugur Cetintemel, Stanley B. Zdonik, Mitch Cherniack, and Michael Stonebraker. Load shedding in a data stream manager. In *VLDB*, 2003.

[TFPL04] Yufei Tao, Christos Faloutsos, Dimitris Papadias, and Bin Liu. Prediction and indexing of moving objects with unknown motion patterns. In *Proc. SIGMOD*, 2004.

[TTPF01] Agma Traina, Caetano Traina, Spiros Papadimitriou, and Christos Faloutsos. Tri-plots: Scalable tools for multidimensional data mining. In *Proc. KDD*, pages 184–193, 2001.

[VG87] D.J̃aques Vaisey and Allen Gersho. Variable block-size image coding. In *Proc. ICASSP*, 1987.

[WFYH03] Haixun Wang, Wei Fan, Philip S. Yu, and Jiawei Han. Mining concept-drifting data streams using ensemble classifiers. In *Proc.ACM SIGKDD*, 2003.

[WG94] Andreas S. Weigend and Neil A. Gerschenfeld. *Time Series Prediction: Forecasting the Future and Understanding the Past*. Addison Wesley, 1994.

[WWYY02] Haixun Wang, Wei Wang, Jiong Yang, and Philip S. Yu. Clustering by pattern similarity in large data sets. In *Proc. SIGMOD*. ACM Press, 2002.

[Yan95] Bin Yang. Projection approximation subspace tracking. *IEEE Trans. Sig. Proc.*, 43(1):95–107, 1995.

[YG03] Yong Yao and Johannes Gehrke. Query processing in sensor networks. In *CIDR*, 2003.

[YHC02] H. Yu, J. Han, and K. Chang. PEBL: Positive example based learning for web page classification using SVM. In *Proc. KDD*, 2002.

[You84] Peter Young. *Recursive Estimation and Time-Series Analysis: An Introduction*. Springer-Verlag, 1984.

[YSJ+00] Byoung-Kee Yi, N.D. Sidiropoulos, Theodore Johnson, H.V. Jagadish, Christos Faloutsos, and Alexandros Biliris. Online data mining for co-evolving time sequences. *Proc. ICDE*, 2000.

[YT01] K. Yamanishi and J. Takeuchi. Discovering outlier filtering rules from unlabeled data. In *Proc. KDD*, 2001.

[ZdZ98] R.A. Zuidwijk and P.M. de Zeeuw. Fast algorithm for directional time-scale analysis using wavelets. In *Proc. SPIE, Wavelet Applications in Signal and Image Processing VI*, volume 3458, 1998.

[ZGTS02] Donghui Zhang, Dimitrios Gunopulos, Vassilis J. Tsotras, and Bernhard Seeger. Temporal aggregation over data streams using multiple granularities. In *Proc. EDBT*, 2002.

[ZHD00] Bin Zhang, Meichun Hsu, and Umeshwar Dayal. K-harmonic means—a spatial clustering algorithm with boosting. In *Proc. TSDM*, pages 31–45, 2000.

[ZK04] Ramin Zabih and Vladimir Kolmogorov. Spatially coherent clustering with graph cuts. In *Proc. CVPR*, 2004.

[ZKPF04] Cui Zhu, Hiroyuki Kitagawa, Spiros Papadimitriou, and Christos Faloutsos. OBE: Outlier by example. In *Proc. PAKDD*, 2004.

[ZMCS04] Xin Zhang, Nikos Mamoulis, David W. Cheung, and Yutao Shou. Fast mining of spatial collocations. In *Proc. KDD*, pages 384–393, 2004.

[ZRL96] Tian Zhang, Raghu Ramakrishnan, and Miron Livny. BIRCH: An efficient data clustering method for very large databases. In *Proc. SIGMOD*, pages 103–114, 1996.

[ZS02] Yunyue Zhu and Dennis Shasha. StatStream: Statistical monitoring of thousands of data streams in real time. In *Proc. VLDB*, 2002.

[ZS03] Yunyue Zhu and Dennis Shasha. Efficient elastic burst detection in data streams. In *Proc. KDD,* 2003.